tox Documentation

Release 3.4.0

holger krekel and others

Sep 20, 2018

Contents

CHAPTER 1

vision: standardize testing in Python

`tox` aims to automate and standardize testing in Python. It is part of a larger vision of easing the packaging, testing and release process of Python software.

CHAPTER 2

What is tox?

tox is a generic virtualenv management and test command line tool you can use for:

- checking your package installs correctly with different Python versions and interpreters
- running your tests in each of the environments, configuring your test tool of choice
- acting as a frontend to Continuous Integration servers, greatly reducing boilerplate and merging CI and shell-based testing.

CHAPTER 3

Basic example

First, install tox with `pip install tox`. Then put basic information about your project and the test environments you want your project to run in into a `tox.ini` file residing right next to your `setup.py` file:

```
# content of: tox.ini , put in same dir as setup.py
[tox]
envlist = py27,py36

[testenv]
deps = pytest       # install pytest in the virtualenv where commands will be executed
commands =
    # whatever extra steps before testing might be necessary
    pytest          # or any other test runner that you might use
```

You can also try generating a `tox.ini` file automatically, by running `tox-quickstart` and then answering a few simple questions.

To sdist-package, install and test your project against Python2.7 and Python3.6, just type:

```
tox
```

and watch things happening (you must have python2.7 and python3.6 installed in your environment otherwise you will see errors). When you run `tox` a second time you'll note that it runs much faster because it keeps track of virtualenv details and will not recreate or re-install dependencies. You also might want to checkout *tox configuration and usage examples* to get some more ideas.

CHAPTER 4

System overview

tox roughly follows the following phases:

1. **configuration:** load `tox.ini` and merge it with options from the command line and the operating system environment variables.
2. **packaging** (optional): create a source distribution of the current project by invoking

   ```
   python setup.py sdist
   ```

   Note that for this operation the same Python environment will be used as the one tox is installed into (therefore you need to make sure that it contains your build dependencies). Skip this step for application projects that don't have a `setup.py`.
3. **environment** - for each tox environment (e.g. `py27`, `py36`) do:

   1. **environment creation**: create a fresh environment, by default virtualenv is used. tox will automatically try to discover a valid Python interpreter version by using the environment name (e.g. `py27` means Python 2.7 and the `basepython` configuration value) and the current operating system `PATH` value. This is created at first run only to be re-used at subsequent runs. If certain aspects of the project change, a re-creation of the environment is automatically triggered. To force the recreation tox can be invoked with `-r`/`--recreate`.

   2. **install** (optional): install the environment dependencies specified inside the `deps` configuration section, and then the earlier packaged source distribution. By default `pip` is used to install packages, however one can customise this via `install_command`. Note `pip` will not update project dependencies (specified either in the `install_requires` or the `extras` section of the `setup.py`) if any version already exists in the virtual environment; therefore we recommend to recreate your environments whenever your project dependencies change.

   3. **commands**: run the specified commands in the specified order. Whenever the exit code of any of them is not zero stop, and mark the environment failed. Note, starting a command with a single dash character means ignore exit code.

6. **report** print out a report of outcomes for each tox environment:

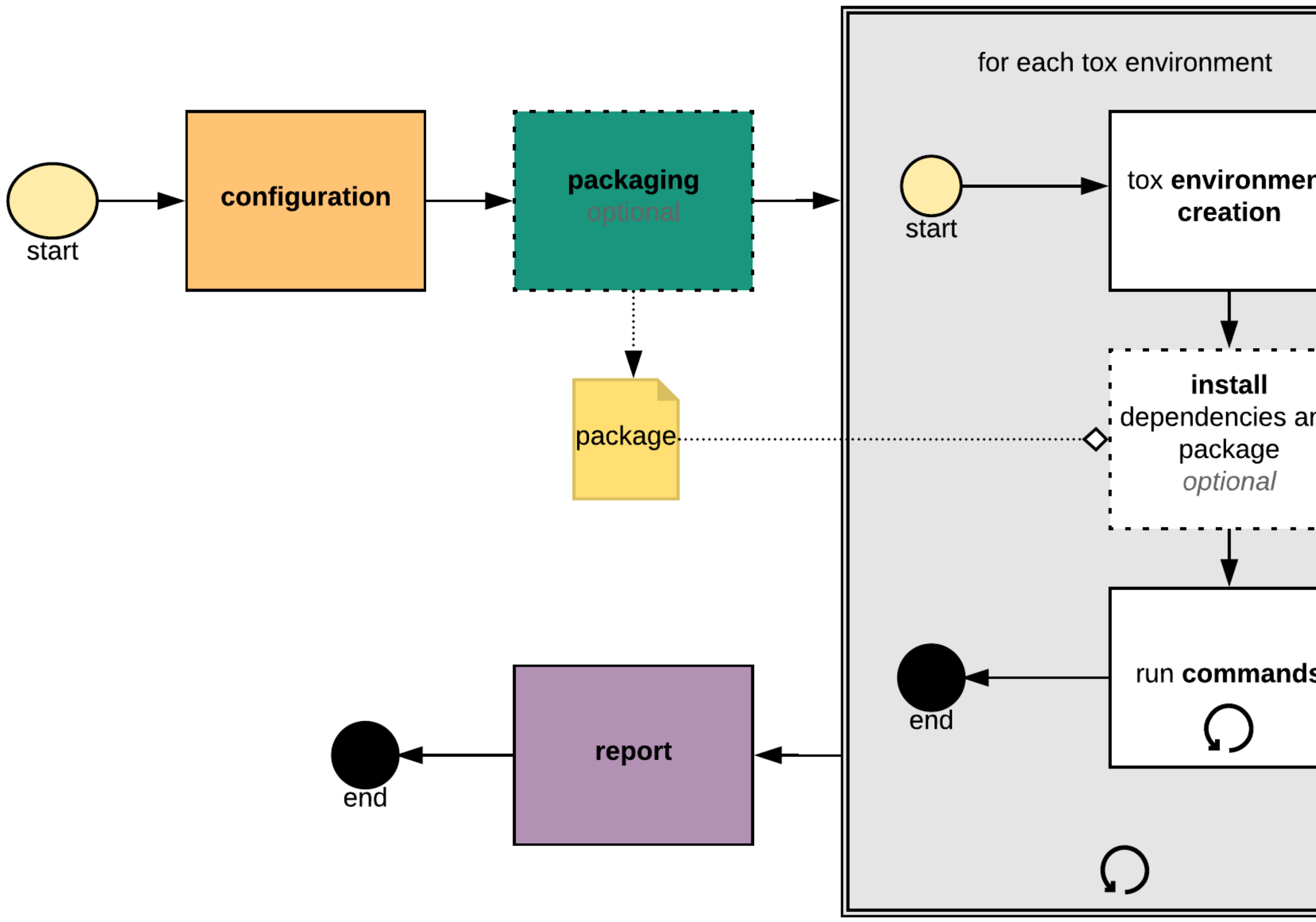

Fig. 1: tox workflow diagram

```
  _______________________ summary _______________________
  py27: commands succeeded
  ERROR:   py36: commands failed

Only if all environments ran successfully tox will return exit code ``0``␣
↪(success). In this
case you'll also see the message ``congratulations :)``.
```

tox will take care of environment isolation for you: it will strip away all operating system environment variables not specified via `passenv`. Furthermore, it will also alter the `PATH` variable so that your commands resolve first and foremost within the current active tox environment. In general all executables in the path are available in `commands`, but tox will emit a warning if it was not explicitly allowed via `whitelist_external`.

CHAPTER 5

Current features

- **automation of tedious Python related test activities**
- **test your Python package against many interpreter and dependency configs**
  - automatic customizable (re)creation of virtualenv test environments
  - installs your `setup.py` based project into each virtual environment
  - test-tool agnostic: runs pytest, nose or unittests in a uniform manner
- *plugin system* to modify tox execution with simple hooks.
- uses pip and setuptools by default. Support for configuring the installer command through `install_command=ARGV`.
- **cross-Python compatible**: CPython-2.7, 3.4 and higher, Jython and pypy.
- **cross-platform**: Windows and Unix style environments
- **integrates with continuous integration servers** like Jenkins (formerly known as Hudson) and helps you to avoid boilerplatish and platform-specific build-step hacks.
- **full interoperability with devpi**: is integrated with and is used for testing in the devpi system, a versatile pypi index server and release managing tool.
- **driven by a simple ini-style config file**
- **documented** *examples* and *configuration*
- **concise reporting** about tool invocations and configuration errors
- **professionally** *supported*
- supports *using different / multiple PyPI index servers*

5.1 tox installation

5.1.1 Install info in a nutshell

Pythons: CPython 2.7 and 3.4 or later, Jython-2.5.1, pypy-1.9ff

Operating systems: Linux, Windows, OSX, Unix

Installer Requirements: setuptools

License: MIT license

git repository: https://github.com/tox-dev/tox

5.1.2 Installation with pip

Use the following command:

```
pip install tox
```

It is fine to install `tox` itself into a virtualenv environment.

5.1.3 Install from Checkout

Consult the GitHub page to get a checkout of the git repository:

https://github.com/tox-dev/tox

and then install in your environment with something like:

```
python setup.py install
```

or just activate your checkout in your environment like this:

```
python setup.py develop
```

so that you can do changes and submit patches.

5.2 tox configuration and usage examples

5.2.1 Basic usage

a simple tox.ini / default environments

Put basic information about your project and the test environments you want your project to run in into a `tox.ini` file that should reside next to your `setup.py` file:

```
# content of: tox.ini , put in same dir as setup.py
[tox]
envlist = py27,py36
[testenv]
deps=pytest # or 'nose' or ...
commands=pytest  # or 'nosetests' or ...
```

To sdist-package, install and test your project, you can now type at the command prompt:

```
tox
```

This will sdist-package your current project, create two virtualenv Environments, install the sdist-package into the environments and run the specified command in each of them. With:

```
tox -e py36
```

you can run restrict the test run to the python3.6 environment.

Available "default" test environments names are:

```
py
py2
py27
py3
py34
py35
py36
py37
py38
jython
pypy
pypy2
pypy27
pypy3
pypy35
```

The environment `py` uses the version of Python used to invoke tox.

However, you can also create your own test environment names, see some of the examples in *examples*.

pyproject.toml tox legacy ini

It's possible to put the tox ini configuration into the `pyproject.toml` file too (if want to avoid an extra file):

```
[build-system]
requires = [ "setuptools >= 35.0.2", "wheel >= 0.29.0"]
build-backend = "setuptools.build_meta"

[tool.tox]
legacy_tox_ini = """
[tox]
envlist = py27,py36

[testenv]
deps = pytest >= 3.0.0, <4
commands = pytest
"""
```

Note that when you define a `pyproject.toml` you must define the `build-requires` section per PEP-518.

specifying a platform

New in version 2.0.

If you want to specify which platform(s) your test environment runs on you can set a platform regular expression like this:

```
[testenv]
platform = linux2|darwin
```

If the expression does not match against `sys.platform` the test environment will be skipped.

whitelisting non-virtualenv commands

New in version 1.5.

Sometimes you may want to use tools not contained in your virtualenv such as `make`, `bash` or others. To avoid warnings you can use the `whitelist_externals` testenv configuration:

```
# content of tox.ini
[testenv]
whitelist_externals = make
                      /bin/bash
```

depending on requirements.txt or defining constraints

New in version 1.6.1.

(experimental) If you have a `requirements.txt` file or a `constraints.txt` file you can add it to your `deps` variable like this:

```
[testenv]
deps = -rrequirements.txt
```

or

```
[testenv]
deps = -cconstraints.txt
```

or

```
[testenv]
deps = -rrequirements.txt -cconstraints.txt
```

All installation commands are executed using `{toxinidir}` (the directory where `tox.ini` resides) as the current working directory. Therefore, the underlying `pip` installation will assume `requirements.txt` or `constraints.txt` to exist at `{toxinidir}/requirements.txt` or `{toxinidir}/contrains.txt`.

This is actually a side effect that all elements of the dependency list is directly passed to `pip`.

For more details on `requirements.txt` files or `constraints.txt` files please see:

- https://pip.pypa.io/en/stable/user_guide/#requirements-files
- https://pip.pypa.io/en/stable/user_guide/#constraints-files

using a different default PyPI url

New in version 0.9.

To install dependencies and packages from a different default PyPI server you can type interactively:

```
tox -i https://pypi.my-alternative-index.org
```

This causes tox to install dependencies and the sdist install step to use the specified url as the index server.

You can cause the same effect by this `tox.ini` content:

```
[tox]
indexserver =
    default = https://pypi.my-alternative-index.org
```

installing dependencies from multiple PyPI servers

New in version 0.9.

You can instrument tox to install dependencies from different PyPI servers, example:

```
[tox]
indexserver =
    DEV = https://mypypiserver.org

[testenv]
deps =
    docutils        # comes from standard PyPI
    :DEV:mypackage  # will be installed from custom "DEV" pypi url
```

This configuration will install `docutils` from the default Python PYPI server and will install the `mypackage` from our `DEV` indexserver, and the respective `https://mypypiserver.org` url. You can override config file settings from the command line like this:

```
tox -i DEV=https://pypi.org/simple  # changes :DEV: package URLs
tox -i https://pypi.org/simple      # changes default
```

further customizing installation

New in version 1.6.

By default tox uses pip to install packages, both the package-under-test and any dependencies you specify in `tox.ini`. You can fully customize tox's install-command through the testenv-specific *`install_command=ARGV`* setting. For instance, to use pip's `--find-links` and `--no-index` options to specify an alternative source for your dependencies:

```
[testenv]
install_command = pip install --pre --find-links https://packages.example.com --no-
↪index {opts} {packages}
```

forcing re-creation of virtual environments

New in version 0.9.

To force tox to recreate a (particular) virtual environment:

```
tox --recreate -e py27
```

would trigger a complete reinstallation of the existing py27 environment (or create it afresh if it doesn't exist).

passing down environment variables

New in version 2.0.

By default tox will only pass the `PATH` environment variable (and on windows `SYSTEMROOT` and `PATHEXT`) from the tox invocation to the test environments. If you want to pass down additional environment variables you can use the `passenv` option:

```ini
[testenv]
passenv = LANG
```

When your test commands execute they will execute with the same LANG setting as the one with which tox was invoked.

setting environment variables

New in version 1.0.

If you need to set an environment variable like `PYTHONPATH` you can use the `setenv` directive:

```ini
[testenv]
setenv = PYTHONPATH = {toxinidir}/subdir
```

When your test commands execute they will execute with a PYTHONPATH setting that will lead Python to also import from the `subdir` below the directory where your `tox.ini` file resides.

special handling of PYTHONHASHSEED

New in version 1.6.2.

By default, tox sets PYTHONHASHSEED for test commands to a random integer generated when `tox` is invoked. This mimics Python's hash randomization enabled by default starting in Python 3.3. To aid in reproducing test failures, tox displays the value of `PYTHONHASHSEED` in the test output.

You can tell tox to use an explicit hash seed value via the `--hashseed` command-line option to `tox`. You can also override the hash seed value per test environment in `tox.ini` as follows:

```ini
[testenv]
setenv = PYTHONHASHSEED = 100
```

If you wish to disable this feature, you can pass the command line option `--hashseed=noset` when `tox` is invoked. You can also disable it from the `tox.ini` by setting `PYTHONHASHSEED = 0` as described above.

Integration with "setup.py test" command

Warning: Integrating tox with `setup.py test` is as of October 2016 discouraged as it breaks packaging/testing approaches used by downstream distributions which expect `setup.py test` to run tests with the invocation interpreter rather than setting up many virtualenvs and installing packages. If you need to define `setup.py test`, you can see how to integrate your eventual test runner with it, here is an example of setup.py test integration with pytest. As the python eco-system rather moves away from using `setup.py` as a tool entry point it's maybe best to not go for any `setup.py test` integration.

Ignoring a command exit code

In some cases, you may want to ignore a command exit code. For example:

```
[testenv:py27]
commands = coverage erase
       {envbindir}/python setup.py develop
       coverage run -p setup.py test
       coverage combine
       - coverage html
       {envbindir}/flake8 loads
```

By using the - prefix, similar to a `make` recipe line, you can ignore the exit code for that command.

Compressing dependency matrix

If you have a large matrix of dependencies, python versions and/or environments you can use *Generative envlist* and *conditional settings* to express that in a concise form:

```
[tox]
envlist = py{27,34,36}-django{15,16}-{sqlite,mysql}

[testenv]
deps =
    django15: Django>=1.5,<1.6
    django16: Django>=1.6,<1.7
    py34-mysql: PyMySQL     ; use if both py34 and mysql are in an env name
    py27,py36: urllib3      ; use if any of py36 or py27 are in an env name
    py{27,36}-sqlite: mock  ; mocking sqlite in python 2.x
```

Prevent symbolic links in virtualenv

By default virtualenv will use symlinks to point to the system's python files, modules, etc. If you want the files to be copied instead, possibly because your filesystem is not capable of handling symbolic links, you can instruct virtualenv to use the "–always-copy" argument meant exactly for that purpose, by setting the `alwayscopy` directive in your environment:

```
[testenv]
alwayscopy = True
```

5.2.2 packaging

Although one can use tox to develop and test applications one of its most popular usage is to help library creators. Libraries need first to be packaged, so then they can be installed inside a virtual environment for testing. To help with this tox implements PEP-517 and PEP-518. This means that by default tox will build source distribution out of source trees. Before running test commands `pip` is used to install the source distribution inside the build environment.

To create a source distribution there are multiple tools out there and with `PEP-517` and `PEP-518` you can easily use your favorite one with tox. Historically tox only supported `setuptools`, and always used the tox host environment to build a source distribution from the source tree. This is still the default behavior. To opt out of this behaviour you need to set isolated builds to true.

setuptools

Using the `pyproject.toml` file at the root folder (alongside `setup.py`) one can specify build requirements.

```
[build-system]
requires = [
    "setuptools >= 35.0.2",
    "setuptools_scm >= 2.0.0, <3"
]
build-backend = "setuptools.build_meta"
```

```
# tox.ini
[tox]
build_isolated = True
```

flit

flit requires `Python 3`, however the generated source distribution can be installed under `python 2`. Furthermore it does not require a `setup.py` file as that information is also added to the `pyproject.toml` file.

```
[build-system]
requires = ["flit >= 1.1"]
build-backend = "flit.buildapi"

[tool.flit.metadata]
module = "package_toml_flit"
author = "Happy Harry"
author-email = "happy@harry.com"
home-page = "https://github.com/happy-harry/is"
```

```
# tox.ini
[tox]
build_isolated = True
```

5.2.3 pytest and tox

It is easy to integrate pytest runs with tox. If you encounter issues, please check if they are *listed as a known issue* and/or use the *support channels*.

Basic example

Assuming the following layout:

```
tox.ini      # see below for content
setup.py     # a classic distutils/setuptools setup.py file
```

and the following `tox.ini` content:

```
[tox]
envlist = py35,py36

[testenv]
```

(continues on next page)

(continued from previous page)

```
deps = pytest                    # PYPI package providing pytest
commands = pytest {posargs} # substitute with tox' positional arguments
```

you can now invoke `tox` in the directory where your `tox.ini` resides. `tox` will sdist-package your project, create two virtualenv environments with the `python3.5` and `python3.6` interpreters, respectively, and will then run the specified test command in each of them.

Extended example: change dir before test and use per-virtualenv tempdir

Assuming the following layout:

```
tox.ini      # see below for content
setup.py     # a classic distutils/setuptools setup.py file
tests        # the directory containing tests
```

and the following `tox.ini` content:

```
[tox]
envlist = py35,py36
[testenv]
changedir=tests
deps=pytest
commands= pytest  --basetemp={envtmpdir}  \ # pytest tempdir setting
                  {posargs}                 # substitute with tox' positional␣
↪arguments
```

you can invoke `tox` in the directory where your `tox.ini` resides. Differently than in the previous example the `pytest` command will be executed with a current working directory set to `tests` and the test run will use the per-virtualenv temporary directory.

Using multiple CPUs for test runs

`pytest` supports distributing tests to multiple processes and hosts through the pytest-xdist plugin. Here is an example configuration to make `tox` use this feature:

```
[testenv]
deps=pytest-xdist
changedir=tests
commands= pytest --basetemp={envtmpdir}  \
                 --confcutdir=..         \
                 -n 3                    \ # use three sub processes
                 {posargs}
```

Known Issues and limitations

Too long filenames. you may encounter "too long filenames" for temporarily created files in your pytest run. Try to not use the "–basetemp" parameter.

installed-versus-checkout version. `pytest` collects test modules on the filesystem and then tries to import them under their fully qualified name. This means that if your test files are importable from somewhere then your `pytest` invocation may end up importing the package from the checkout directory rather than the installed package.

This issue may be characterised by pytest test-collection error messages, in python 3.x environments, that look like:

```
import file mismatch:
imported module 'myproj.foo.tests.test_foo' has this __file__ attribute:
  /home/myuser/repos/myproj/build/lib/myproj/foo/tests/test_foo.py
which is not the same as the test file we want to collect:
  /home/myuser/repos/myproj/myproj/foo/tests/test_foo.py
HINT: remove __pycache__ / .pyc files and/or use a unique basename for your test file␣
↪modules
```

There are a few ways to prevent this.

With installed tests (the tests packages are known to `setup.py`), a safe and explicit option is to give the explicit path `{envsitepackagesdir}/mypkg` to pytest. Alternatively, it is possible to use `changedir` so that checked-out files are outside the import path, then pass `--pyargs mypkg` to pytest.

With tests that won't be installed, the simplest way to run them against your installed package is to avoid `__init__.py` files in test directories; pytest will still find and import them by adding their parent directory to `sys.path` but they won't be copied to other places or be found by Python's import system outside of pytest.

5.2.4 unittest2, discover and tox

Running unittests with 'discover'

The discover project allows to discover and run unittests and we can easily integrate it in a `tox` run. As an example, perform a checkout of Pygments:

```
hg clone https://bitbucket.org/birkenfeld/pygments-main
```

and add the following `tox.ini` to it:

```
[tox]
envlist = py27,py35,py36

[testenv]
changedir = tests
commands = discover
deps = discover
```

If you now invoke `tox` you will see the creation of three virtual environments and a unittest-run performed in each of them.

Running unittest2 and sphinx tests in one go

Michael Foord has contributed a `tox.ini` file that allows you to run all tests for his mock project, including some sphinx-based doctests. If you checkout its repository with:

```
git clone https://github.com/testing-cabal/mock.git
```

The checkout has a tox.ini file that looks like this:

```
[tox]
envlist = py27,py34,py35,py36

[testenv]
deps = unittest2
```

(continues on next page)

(continued from previous page)

```
commands = unit2 discover []

[testenv:py36]
commands =
    unit2 discover []
    sphinx-build -b doctest docs html
    sphinx-build docs html
deps =
    unittest2
    sphinx

[testenv:py27]
commands =
    unit2 discover []
    sphinx-build -b doctest docs html
    sphinx-build docs html
deps =
    unittest2
    sphinx
```

mock uses unittest2 to run the tests. Invoking `tox` starts test discovery by executing the `unit2 discover` commands on Python 2.7, 3.4, 3.5 and 3.6 respectively. Against Python3.6 and Python2.7 it will additionally run sphinx-mediated doctests. If building the docs fails, due to a reST error, or any of the doctests fails, it will be reported by the tox run.

The `[]` parentheses in the commands provide *interactive shell substitution* which means you can e.g. type:

```
tox -- -f -s SOMEPATH
```

which will ultimately invoke:

```
unit2 discover -f -s SOMEPATH
```

in each of the environments. This allows you to customize test discovery in your `tox` runs.

5.2.5 nose and tox

It is easy to integrate nosetests runs with tox. For starters here is a simple `tox.ini` config to configure your project for running with nose:

Basic nosetests example

Assuming the following layout:

```
tox.ini      # see below for content
setup.py     # a classic distutils/setuptools setup.py file
```

and the following `tox.ini` content:

```
[testenv]
deps=nose
commands= nosetests [] # substitute with tox' positional arguments
```

you can invoke `tox` in the directory where your `tox.ini` resides. `tox` will sdist-package your project create two virtualenv environments with the `python2.7` and `python3.6` interpreters, respectively, and will then run the specified test command.

More examples?

You can use and combine other features of `tox` with your tox runs, e.g. *Integrating "sphinx" documentation checks*. If you figure out some particular configurations for nose/tox interactions please submit them.

Also you might want to checkout *General tips and tricks*.

5.2.6 Generate documentation

It's possible to generate the projects documentation with tox itself. The advantage of this path is that now generating the documentation can be part of the CI, and whenever any validations/checks/operations fail while generating the documentation you'll catch it within tox.

Sphinx

No need to use the cryptic make file to generate a sphinx documentation. One can use tox to ensure all right dependencies are available within a virtual environment, and even specify the python version needed to perform the build. For example if the sphinx file structure is under the `doc` folder the following configuration will generate the documentation under `{toxworkdir}/docs_out` and print out a link to the generated documentation:

```
[testenv:docs]
description = invoke sphinx-build to build the HTML docs
basepython = python3.7
deps = sphinx >= 1.7.5, < 2
commands = sphinx-build -d "{toxworkdir}/docs_doctree" doc "{toxworkdir}/docs_out" --
↪color -W -bhtml {posargs}
           python -c 'import pathlib; print("documentation available under file://\
↪{0\}".format(pathlib.Path(r"{toxworkdir}") / "docs_out" / "index.html"))'
```

Note here we say we also require python 3.7, allowing us to use f-strings within the sphinx `conf.py`. Now one can specify a separate test environment that will validate that the links are correct.

5.2.7 mkdocs

Define one environment to write/generate the documentation, and another to deploy it. Use the config substitution logic to avoid defining dependencies multiple time:

```
[testenv:docs]
description = Run a development server for working on documentation
basepython = python3.7
deps = mkdocs >= 1.7.5, < 2
       mkdocs-material
commands = mkdocs build --clean
           python -c 'print("###### Starting local server. Press Control+C to stop␣
↪server ######")'
           mkdocs serve -a localhost:8080

[testenv:docs-deploy]
description = built fresh docs and deploy them
```

(continues on next page)

(continued from previous page)

```
deps = {[testenv:docs]deps}
basepython = {[testenv:docs]basepython}
commands = mkdocs gh-deploy --clean
```

5.2.8 General tips and tricks

Interactively passing positional arguments

If you invoke `tox` like this:

```
tox -- -x tests/test_something.py
```

the arguments after the `--` will be substituted everywhere where you specify `{posargs}` in your test commands, for example using `pytest`:

```
[testenv]  # or testenv:NAME section of your tox.ini
commands = pytest {posargs}
```

or using `nosetests`:

```
[testenv]
commands = nosetests {posargs}
```

the above `tox` invocation will trigger the test runners to stop after the first failure and to only run a particular test file.

You can specify defaults for the positional arguments using this syntax:

```
[testenv]
commands = nosetests {posargs:--with-coverage}
```

Integrating "sphinx" documentation checks

In a `testenv` environment you can specify any command and thus you can easily integrate sphinx documentation integrity during a tox test run. Here is an example `tox.ini` configuration:

```
[testenv:docs]
changedir = doc
deps = sphinx
commands = sphinx-build -W -b html -d {envtmpdir}/doctrees . {envtmpdir}/html
```

This will create a dedicated `docs` virtual environment and install the `sphinx` dependency which itself will install the `sphinx-build` tool which you can then use as a test command. Note that sphinx output is redirected to the virtualenv environment temporary directory to prevent sphinx from caching results between runs.

You can now call:

```
tox
```

which will make the sphinx tests part of your test run.

Selecting one or more environments to run tests against

Using the `-e ENV[,ENV36,...]` option you explicitly list the environments where you want to run tests against. For example, given the previous sphinx example you may call:

```
tox -e docs
```

which will make `tox` only manage the `docs` environment and call its test commands. You may specify more than one environment like this:

```
tox -e py27,py36
```

which would run the commands of the `py27` and `py36` testenvironments respectively. The special value `ALL` selects all environments.

You can also specify an environment list in your `tox.ini`:

```
[tox]
envlist = py27,py36
```

or override it from the command line or from the environment variable `TOXENV`:

```
export TOXENV=py27,py36 # in bash style shells
```

Access package artifacts between multiple tox-runs

If you have multiple projects using tox you can make use of a `distshare` directory where `tox` will copy in sdist-packages so that another tox run can find the "latest" dependency. This feature allows to test a package against an unreleased development version or even an uncommitted version on your own machine.

By default, `{homedir}/.tox/distshare` will be used for copying in and copying out artifacts (i.e. Python packages).

For project `two` to depend on the `one` package you use the following entry:

```
# example two/tox.ini
[testenv]
deps = {distshare}/one-*.zip  # install latest package from "one" project
```

That's all. tox running on project `one` will copy the sdist-package into the `distshare` directory after which a `tox` run on project `two` will grab it because `deps` contain an entry with the `one-*.zip` pattern. If there is more than one matching package the highest version will be taken. `tox` uses verlib to compare version strings which must be compliant with **PEP 386**.

If you want to use this with Jenkins, also checkout the *Access package artifacts between Jenkins jobs*.

basepython defaults, overriding

For any `pyXY` test environment name the underlying `pythonX.Y` executable will be searched in your system `PATH`. Similarly, for `jython` and `pypy` the respective `jython` and `pypy-c` names will be looked for. The executable must exist in order to successfully create *virtualenv* environments. On Windows a `pythonX.Y` named executable will be searched in typical default locations using the `C:\PythonX.Y\python.exe` pattern.

All other targets will use the system `python` instead. You can override any of the default settings by defining the `basepython` variable in a specific test environment section, for example:

```
[testenv:docs]
basepython = python2.7
```

Avoiding expensive sdist

Some projects are large enough that running an sdist, followed by an install every time can be prohibitively costly. To solve this, there are two different options you can add to the `tox` section. First, you can simply ask tox to please not make an sdist:

```
[tox]
skipsdist=True
```

If you do this, your local software package will not be installed into the virtualenv. You should probably be okay with that, or take steps to deal with it in your commands section:

```
[testenv]
commands = python setup.py develop
           pytest
```

Running `setup.py develop` is a common enough model that it has its own option:

```
[testenv]
usedevelop=True
```

And a corresponding command line option `--develop`, which will set `skipsdist` to True and then perform the `setup.py develop` step at the place where `tox` normally performs the installation of the sdist. Specifically, it actually runs `pip install -e .` behind the scenes, which itself calls `setup.py develop`.

There is an optimization coded in to not bother re-running the command if `$projectname.egg-info` is newer than `setup.py` or `setup.cfg`.

Understanding `InvocationError` exit codes

When a command (defined by `commands =` in `tox.ini`) fails, it has a non-zero exit code, and an `InvocationError` exception is raised by `tox`:

```
ERROR: InvocationError for command
       '<command defined in tox.ini>' (exited with code 1)
```

If the command starts with `pytest` or `python setup.py test` for instance, then the pytest exit codes are relevant.

On unix systems, there are some rather common exit codes. This is why for exit codes larger than 128, if a signal with number equal to `<exit code> - 128` is found in the `signal` module, an additional hint is given:

```
ERROR: InvocationError for command
       '<command>' (exited with code 139)
Note: this might indicate a fatal error signal (139 - 128 = 11: SIGSEGV)
```

where `<command>` is the command defined in `tox.ini`, with quotes removed.

The signal numbers (e.g. 11 for a segmentation fault) can be found in the "Standard signals" section of the signal man page. Their meaning is described in POSIX signals.

Beware that programs may issue custom exit codes with any value, so their documentation should be consulted.

Sometimes, no exit code is given at all. An example may be found in pytest-qt issue #170, where Qt was calling `abort()` instead of `exit()`.

See also:

Ignoring a command exit code.

5.2.9 Using tox with the Jenkins Integration Server

Using Jenkins multi-configuration jobs

The Jenkins continuous integration server allows to define "jobs" with "build steps" which can be test invocations. If you *install* `tox` on your default Python installation on each Jenkins slave, you can easily create a Jenkins multi-configuration job that will drive your tox runs from the CI-server side, using these steps:

- install the Python plugin for Jenkins under "manage jenkins"
- create a "multi-configuration" job, give it a name of your choice
- configure your repository so that Jenkins can pull it
- (optional) configure multiple nodes so that tox-runs are performed on multiple hosts
- configure `axes` by using *TOXENV* as an axis name and as values provide space-separated test environment names you want Jenkins/tox to execute.
- add a **Python-build step** with this content (see also next example):

  ```
  import tox

  os.chdir(os.getenv("WORKSPACE"))
  tox.cmdline()  # environment is selected by ``TOXENV`` env variable
  ```

- check `Publish JUnit test result report` and enter `**/junit-*.xml` as the pattern so that Jenkins collects test results in the JUnit XML format.

The last point requires that your test command creates JunitXML files, for example with `pytest` it is done like this:

```
[testenv]
commands = pytest --junitxml=junit-{envname}.xml
```

zero-installation for slaves

Note: This feature is broken currently because "toxbootstrap.py" has been removed. Please file an issue if you'd like to see it back.

If you manage many Jenkins slaves and want to use the latest officially released tox (or latest development version) and want to skip manually installing `tox` then substitute the above **Python build step** code with this:

```
import urllib, os

url = "https://bitbucket.org/hpk42/tox/raw/default/toxbootstrap.py"
# os.environ['USETOXDEV']="1"  # use tox dev version
d = dict(__file__="toxbootstrap.py")
exec urllib.urlopen(url).read() in d
d["cmdline"](["--recreate"])
```

The downloaded `toxbootstrap.py` file downloads all necessary files to install `tox` in a virtual sub environment. Notes:

- uncomment the line containing `USETOXDEV` to use the latest development-release version of tox instead of the latest released version.
- adapt the options in the last line as needed (the example code will cause tox to reinstall all virtual environments all the time which is often what one wants in CI server contexts)

Integrating "sphinx" documentation checks in a Jenkins job

If you are using a multi-configuration Jenkins job which collects JUnit Test results you will run into problems using the previous method of running the sphinx-build command because it will not generate JUnit results. To accommodate this issue one solution is to have `pytest` wrap the sphinx-checks and create a JUnit result file which wraps the result of calling sphinx-build. Here is an example:

1. create a `docs` environment in your `tox.ini` file like this:

```
[testenv:docs]
basepython = python
changedir = doc # or wherever you keep your sphinx-docs
deps = sphinx
       py
commands = pytest --tb=line -v --junitxml=junit-{envname}.xml check_sphinx.py
```

2. create a `doc/check_sphinx.py` file like this:

```
import py
import subprocess


def test_linkcheck(tmpdir):
    doctrees = tmpdir.join("doctrees")
    htmldir = tmpdir.join("html")
    subprocess.check_call(
        ["sphinx-build", "-W", "-blinkcheck", "-d", str(doctrees), ".",
↪str(htmldir)]
    )


def test_build_docs(tmpdir):
    doctrees = tmpdir.join("doctrees")
    htmldir = tmpdir.join("html")
    subprocess.check_call(
        ["sphinx-build", "-W", "-bhtml", "-d", str(doctrees), ".",
↪str(htmldir)]
    )
```

3. run `tox -e docs` and then you may integrate this environment along with your other environments into Jenkins.

Note that `pytest` is only installed into the docs environment and does not need to be in use or installed with any other environment.

Access package artifacts between Jenkins jobs

In an extension to *Access package artifacts between multiple tox-runs* you can also configure Jenkins jobs to access each others artifacts. `tox` uses the `distshare` directory to access artifacts and in a Jenkins context (detected via existence of the environment variable `HUDSON_URL`); it defaults to to `{toxworkdir}/distshare`.

This means that each workspace will have its own `distshare` directory and we need to configure Jenkins to perform artifact copying. The recommend way to do this is to install the Jenkins Copy Artifact plugin and for each job which "receives" artifacts you add a **Copy artifacts from another project** build step using roughly this configuration:

```
Project-name: name of the other (tox-managed) job you want the artifact from
Artifacts to copy: .tox/dist/*.zip   # where tox jobs create artifacts
Target directory: .tox/distshare     # where we want it to appear for us
Flatten Directories: CHECK           # create no subdir-structure
```

You also need to configure the "other" job to archive artifacts; This is done by checking `Archive the artifacts` and entering:

```
Files to archive: .tox/dist/*.zip
```

So our "other" job will create an sdist-package artifact and the "copy-artifacts" plugin will copy it to our `distshare` area. Now everything proceeds as *Access package artifacts between multiple tox-runs* shows it.

So if you are using defaults you can re-use and debug exactly the same `tox.ini` file and make use of automatic sharing of your artifacts between runs or Jenkins jobs.

Avoiding the "path too long" error with long shebang lines

When using `tox` on a Jenkins instance, there may be a scenario where `tox` can not invoke `pip` because the shebang (Unix) line is too long. Some systems only support a limited amount of characters for an interpreter directive (e.x. Linux as a limit of 128). There are two methods to workaround this issue:

1. Invoke `tox` with the `--workdir` option which tells `tox` to use a specific directory for its virtual environments. Using a unique and short path can prevent this issue.
2. Use the environment variable `TOX_LIMITED_SHEBANG` to deal with environments with interpreter directive limitations (consult *Handle interpreter directives with long lengths* for more information).

Running tox environments in parallel

Jenkins has parallel stages allowing you to run commands in parallel, however tox package building it is not parallel safe. Use the `--parallel--safe-build` flag to enable parallel safe builds (this will generate unique folder names for `distdir`, `ditshare` and `log`. Here's a generic stage definition demonstrating how to use this inside Jenkins:

```
stage('run tox envs') {
  steps {
    script {
      def envs = sh(returnStdout: true, script: "tox -l").trim().split('\n')
      def cmds = envs.collectEntries({ tox_env ->
        [tox_env, {
          sh "tox --parallel--safe-build -vve $tox_env"
        }]
      })
      parallel(cmds)
    }
```

(continues on next page)

(continued from previous page)

```
    }
}
```

5.2.10 Development environment

tox can be used for just preparing different virtual environments required by a project.

This feature can be used by deployment tools when preparing deployed project environments. It can also be used for setting up normalized project development environments and thus help reduce the risk of different team members using mismatched development environments.

Here are some examples illustrating how to set up a project's development environment using tox. For illustration purposes, let us call the development environment `devenv`.

Example 1: Basic scenario

Step 1 - Configure the development environment

First, we prepare the tox configuration for our development environment by defining a `[testenv:devenv]` section in the project's `tox.ini` configuration file:

```
[testenv:devenv]
envdir = devenv
basepython = python2.7
usedevelop = True
```

In it we state:

- what directory to locate the environment in,
- what Python executable to use in the environment,
- that our project should be installed into the environment using `setup.py develop`, as opposed to building and installing its source distribution using `setup.py install`.

Actually, we can configure a lot more, and these are only the required settings. For example, we can add the following to our configuration, telling tox not to reuse `commands` or `deps` settings from the base `[testenv]` configuration:

```
[testenv:devenv]
commands =
deps =
```

Step 2 - Create the development environment

Once the `[testenv:devenv]` configuration section has been defined, we create the actual development environment by running the following:

```
tox -e devenv
```

This creates the environment at the path specified by the environment's `envdir` configuration value.

Example 2: A more complex scenario

Let us say we want our project development environment to:

- be located in the `devenv` directory,
- use Python executable `python2.7`,
- pull packages from `requirements.txt`, located in the same directory as `tox.ini`.

Here is an example configuration for the described scenario:

```
[testenv:devenv]
envdir = devenv
basepython = python2.7
usedevelop = True
deps = -rrequirements.txt
```

5.2.11 Platform specification

Basic multi-platform example

Assuming the following layout:

```
tox.ini      # see below for content
setup.py     # a classic distutils/setuptools setup.py file
```

and the following `tox.ini` content:

```
[tox]
# platform specification support is available since version 2.0
minversion = 2.0
envlist = py{27,36}-{mylinux,mymacos,mywindows}

[testenv]
# environment will be skipped if regular expression does not match against the sys.
↪platform string
platform = mylinux: linux
           mymacos: darwin
           mywindows: win32

# you can specify dependencies and their versions based on platform filtered␣
↪environments
deps = mylinux,mymacos: py==1.4.32
       mywindows: py==1.4.30

# upon tox invocation you will be greeted according to your platform
commands=
   mylinux: python -c 'print("Hello, Linus!")'
   mymacos: python -c 'print("Hello, Steve!")'
   mywindows: python -c 'print("Hello, Bill!")'
```

you can invoke `tox` in the directory where your `tox.ini` resides. `tox` creates two virtualenv environments with the `python2.7` and `python3.6` interpreters, respectively, and will then run the specified command according to platform you invoke `tox` at.

5.3 tox configuration specification

tox supports at the moment three locations for specifying the configuration, in the following priority order:

1. `pyproject.toml`,
2. `tox.ini`,
3. `setup.cfg`.

As far as the configuration format at the moment we only support standard ConfigParser "ini-style" format (there is a plan to add a pure TOML one soon). `tox.ini` and `setup.cfg` are such files. Note that `setup.cfg` requires the content to be under the `tox:tox` section. `pyproject.toml` on the other hand is in TOML format. However, one can inline the *ini-style* format under the `tool.tox.legacy_tox_ini` key as a multi-line string.

Below you find the specification for the *ini-style* format, but you might want to skim some *tox configuration and usage examples* first and use this page as a reference.

5.3.1 tox global settings

List of optional global options:

```
[tox]
# minimally required tox version
minversion=ver
# tox working directory, defaults to {toxinidir}/.tox
toxworkdir=path
# defaults to {toxinidir}
setupdir=path
# defaults to {toxworkdir}/dist
distdir=path
# (DEPRECATED) defaults to {homedir}/.tox/distshare
distshare=path
# defaults to the list of all environments
envlist=ENVLIST
# bool: defaults to False
skipsdist=False
```

`tox` autodetects if it is running in a Jenkins context (by checking for existence of the `JENKINS_URL` environment variable) and will first lookup global tox settings in this section:

```
[tox:jenkins]
commands = ...  # override [tox] settings for the jenkins context
# note: for jenkins distshare defaults to ``{toxworkdir}/distshare`` (DEPRECATED)
```

`skip_missing_interpreters=config|true|false`

New in version 1.7.2.

When skip missing interpreters is `true` will force `tox` to return success even if some of the specified environments were missing. This is useful for some CI systems or running on a developer box, where you might only have a subset of all your supported interpreters installed but don't want to mark the build as failed because of it. As expected, the command line switch always overrides this setting if passed on the invocation. Setting it to `config` means that the value is read from the config file (default is `false`). **Default:** `config`

`envlist=CSV`

Determining the environment list that `tox` is to operate on happens in this order (if any is found, no further lookups are made):

- command line option `-eENVLIST`
- environment variable `TOXENV`
- `tox.ini` file's `envlist`

New in version 3.4.0: What tox environments are ran during the tox invocation can be further filtered via the operating system environment variable `TOX_SKIP_ENV` regular expression (e.g. `py27.*` means **don't** evaluate environments that start with the key `py27`). Skipped environments will be logged at level two verbosity level.

`ignore_basepython_conflict=True|False(default)`

New in version 3.1.0.

tox allows setting the python version for an environment via the `basepython` setting. If that's not set tox can set a default value from the environment name (e.g. `py37` implies Python 3.7). Matching up the python version with the environment name has became expected at this point, leading to surprises when some configs don't do so. To help with sanity of users a warning will be emitted whenever the environment name version does not matches up with this expectation. In a future version of tox, this warning will become an error.

Furthermore, we allow hard enforcing this rule (and bypassing the warning) by setting this flag to `True`. In such cases we ignore the `basepython` and instead always use the base python implied from the Python name. This allows you to configure `basepython` in the global testenv without affecting environments that have implied base python versions.

`requires=LIST`

New in version 3.2.0.

Specify python packages that need to exist alongside the tox installation for the tox build to be able to start. Use this to specify plugin requirements and build dependencies.

```
[tox]
requires = setuptools >= 30.0.0
           py
```

`isolated_build=True|False(default)`

New in version 3.3.0.

Activate isolated build environment. tox will use a virtual environment to build a source distribution from the source tree. For build tools and arguments use the `pyproject.toml` file as specified in PEP-517 and PEP-518. To specify the virtual environment Python version define use the `isolated_build_env` config section.

`isolated_build_env=str`

New in version 3.3.0.

Name of the virtual environment used to create a source distribution from the source tree. By **default ''.package''** is used.

5.3.2 Virtualenv test environment settings

Test environments are defined by a:

```
[testenv:NAME]
commands = ...
```

section. The `NAME` will be the name of the virtual environment. Defaults for each setting in this section are looked up in the:

```
[testenv]
commands = ...
```

testenvironment default section.

Complete list of settings that you can put into `testenv*` sections:

`basepython=NAME-OR-PATH`
Name or path to a Python interpreter which will be used for creating the virtual environment, this determines in practice the python for what we'll create a virtual isolated environment. Use this to specify the python version for a tox environment. If not specified the virtual environments factors (e.g. name part) wil be used to automatically set one. E.g. `py37` means `python3.7`, `py3` means `python3` and `py` means `python`.

Changed in version 3.1: After resolving this value if the interpreter reports back a different version number than implied from the name a warning will be printed by default. However, if `ignore_basepython_conflict` is set, the value is ignored and we force the `basepython` implied from the factor name.

`commands=ARGVLIST`
The commands to be called for testing. Only execute if `commands_pre` succeed.

Each line is interpreted as one command; however a command can be split over multiple lines by ending the line with the \ character.

Commands will execute one by one in sequential fashion until one of them fails (their exit code is non-zero) or all of them succeed. The exit code of a command may be ignored (meaning they are always considered successful) by prefixing the command with a dash (`-`) - this is similar to how `make` recipe lines work. The outcome of the environment is considered successful only if all commands (these + setup + teardown) succeeded (exit code ignored via the `-` or success exit code value of zero).

> **Note** the virtual environment binary path (the `bin` folder within) is prepended to the os `PATH`, meaning commands will first try to resolve to an executable from within the virtual environment, and only after that outside of it. Therefore `python` translates as the virtual environments `python` (having the same runtime version as the `basepython`), and `pip` translates as the virtual environments `pip`.

`commands_pre=ARGVLIST`
Commands to run before running the `commands`. All evaluation and configuration logic applies from `commands`.

`commands_post=ARGVLIST`
Commands to run after running the `commands`. Execute regardless of the outcome of both `commands` and `commands_pre`. All evaluation and configuration logic applies from `commands`.

`install_command=ARGV`
New in version 1.6.

Determines the command used for installing packages into the virtual environment; both the package under test and its dependencies (defined with `deps`). Must contain the substitution key `{packages}` which will be replaced by the package(s) to install. You should also accept `{opts}` if you are using pip – it will contain index server options such as `--pre` (configured as `pip_pre`) and potentially index-options from the deprecated *`indexserver`* option.

default:

```
python -m pip install {opts} {packages}
```

`list_dependencies_command`
New in version 2.4.

The `list_dependencies_command` setting is used for listing the packages installed into the virtual environment.

default:

```
pip freeze
```

`ignore_errors=True|False(default)`

New in version 2.0.

If `True`, a non-zero exit code from one command will be ignored and further commands will be executed (which was the default behavior in tox < 2.0). If `False` (the default), then a non-zero exit code from one command will abort execution of commands for that environment.

It may be helpful to note that this setting is analogous to the `-i` or `ignore-errors` option of GNU Make. A similar name was chosen to reflect the similarity in function.

Note that in tox 2.0, the default behavior of tox with respect to treating errors from commands changed. tox < 2.0 would ignore errors by default. tox >= 2.0 will abort on an error by default, which is safer and more typical of CI and command execution tools, as it doesn't make sense to run tests if installing some prerequisite failed and it doesn't make sense to try to deploy if tests failed.

`pip_pre=True|False(default)`

New in version 1.9.

If `True`, adds `--pre` to the `opts` passed to `install_command`. If `install_command` uses pip, this will cause it to install the latest available pre-release of any dependencies without a specified version. If `False` (the default), pip will only install final releases of unpinned dependencies.

Passing the `--pre` command-line option to tox will force this to `True` for all testenvs.

Don't set this option if your `install_command` does not use pip.

`whitelist_externals=MULTI-LINE-LIST`

each line specifies a command name (in glob-style pattern format) which can be used in the `commands` section without triggering a "not installed in virtualenv" warning. Example: if you use the unix `make` for running tests you can list `whitelist_externals=make` or `whitelist_externals=/usr/bin/make` if you want more precision. If you don't want tox to issue a warning in any case, just use `whitelist_externals=*` which will match all commands (not recommended).

`changedir=path`

change to this working directory when executing the test command.

default: `{toxinidir}`

`deps=MULTI-LINE-LIST`

Test-specific dependencies - to be installed into the environment prior to project package installation. Each line defines a dependency, which will be passed to the installer command for processing (see *`indexserver`*). Each line specifies a file, a URL or a package name. You can additionally specify an *`indexserver`* to use for installing this dependency but this functionality is deprecated since tox-2.3. All derived dependencies (deps required by the dep) will then be retrieved from the specified indexserver:

```
[tox]
indexserver =
    myindexserver = https://myindexserver.example.com/simple

[testenv]
deps = :myindexserver:pkg
```

(Experimentally introduced in 1.6.1) all installer commands are executed using the `{toxinidir}` as the current working directory.

platform=REGEX

New in version 2.0.

A testenv can define a new `platform` setting as a regular expression. If a non-empty expression is defined and does not match against the `sys.platform` string the test environment will be skipped.

setenv=MULTI-LINE-LIST

New in version 0.9.

Each line contains a NAME=VALUE environment variable setting which will be used for all test command invocations as well as for installing the sdist package into a virtual environment.

Notice that when updating a path variable, you can consider the use of variable substitution for the current value and to handle path separator.

```
[testenv]
setenv   =
    PYTHONPATH = {env:PYTHONPATH}{:}{toxinidir}
```

passenv=SPACE-SEPARATED-GLOBNAMES

New in version 2.0.

A list of wildcard environment variable names which shall be copied from the tox invocation environment to the test environment when executing test commands. If a specified environment variable doesn't exist in the tox invocation environment it is ignored. You can use `*` and `?` to match multiple environment variables with one name.

Some variables are always passed through to ensure the basic functionality of standard library functions or tooling like pip:

- passed through on all platforms: `PATH`, `LANG`, `LANGUAGE`, `LD_LIBRARY_PATH`, `PIP_INDEX_URL`
- **Windows: SYSTEMDRIVE, SYSTEMROOT, PATHEXT, TEMP, TMP** `NUMBER_OF_PROCESSORS`, `USERPROFILE`, `MSYSTEM`
- Others (e.g. UNIX, macOS): `TMPDIR`

You can override these variables with the `setenv` option.

If defined the `TOX_TESTENV_PASSENV` environment variable (in the tox invocation environment) can define additional space-separated variable names that are to be passed down to the test command environment.

Changed in version 2.7: `PYTHONPATH` will be passed down if explicitly defined. If `PYTHONPATH` exists in the host environment but is **not** declared in `passenv` a warning will be emitted.

recreate=True|False(default)

Always recreate virtual environment if this option is True.

downloadcache=path

IGNORED – Since pip-8 has caching by default this option is now ignored. Please remove it from your configs as a future tox version might bark on it.

sitepackages=True|False

Set to `True` if you want to create virtual environments that also have access to globally installed packages.

default: False, meaning that virtualenvs will be created without inheriting the global site packages.

alwayscopy=True|False

Set to `True` if you want virtualenv to always copy files rather than symlinking.

This is useful for situations where hardlinks don't work (e.g. running in VMS with Windows guests).

default: False, meaning that virtualenvs will make use of symbolic links.

args_are_paths=BOOL
Treat positional arguments passed to `tox` as file system paths and - if they exist on the filesystem - rewrite them according to the `changedir`.

default: True (due to the exists-on-filesystem check it's usually safe to try rewriting).

envtmpdir=path
Defines a temporary directory for the virtualenv which will be cleared each time before the group of test commands is invoked.

default: `{envdir}/tmp`

envlogdir=path
Defines a directory for logging where tox will put logs of tool invocation.

default: `{envdir}/log`

indexserver
New in version 0.9.

(DEPRECATED, will be removed in a future version) Multi-line `name = URL` definitions of python package servers. Dependencies can specify using a specified index server through the `:indexservername:depname` pattern. The `default` indexserver definition determines where unscoped dependencies and the sdist install installs from. Example:

```
[tox]
indexserver =
    default = https://mypypi.org
```

will make tox install all dependencies from this PYPI index server (including when installing the project sdist package).

envdir
New in version 1.5.

User can set specific path for environment. If path would not be absolute it would be treated as relative to `{toxinidir}`.

default: `{toxworkdir}/{envname}`

usedevelop=BOOL
New in version 1.6.

Install the current package in development mode with "setup.py develop" instead of installing from the `sdist` package. (This uses pip's `-e` option, so should be avoided if you've specified a custom `install_command` that does not support `-e`).

default: `False`

skip_install=BOOL
New in version 1.9.

Do not install the current package. This can be used when you need the virtualenv management but do not want to install the current package into that environment.

default: `False`

ignore_outcome=BOOL
New in version 2.2.

If set to True a failing result of this testenv will not make tox fail, only a warning will be produced.

default: `False`

extras=MULTI-LINE-LIST
New in version 2.4.

A list of "extras" to be installed with the sdist or develop install. For example, `extras = testing` is equivalent to `[testing]` in a `pip install` command.

description=SINGLE-LINE-TEXT
A short description of the environment, this will be used to explain the environment to the user upon listing environments for the command line with any level of verbosity higher than zero. **default**: empty string

5.3.3 Substitutions

Any `key=value` setting in an ini-file can make use of value substitution through the `{...}` string-substitution pattern.

You can escape curly braces with the \ character if you need them, for example:

```
commands = echo "\{posargs\}" = {posargs}
```

Globally available substitutions

{toxinidir} the directory where tox.ini is located

{toxworkdir} the directory where virtual environments are created and sub directories for packaging reside.

{homedir} the user-home directory path.

{distdir} the directory where sdist-packages will be created in

{distshare} (DEPRECATED) the directory where sdist-packages will be copied to so that they may be accessed by other processes or tox runs.

{:} OS-specific path separator (**:** os *nix family, **;** on Windows). May be used in `setenv`, when target variable is path variable (e.g. PATH or PYTHONPATH).

substitutions for virtualenv-related sections

{envname} the name of the virtual environment

{envpython} path to the virtual Python interpreter

{envdir} directory of the virtualenv hierarchy

{envbindir} directory where executables are located

{envsitepackagesdir} directory where packages are installed. Note that architecture-specific files may appear in a different directory.

{envtmpdir} the environment temporary directory

{envlogdir} the environment log directory

environment variable substitutions

If you specify a substitution string like this:

```
{env:KEY}
```

then the value will be retrieved as `os.environ['KEY']` and raise an Error if the environment variable does not exist.

environment variable substitutions with default values

If you specify a substitution string like this:

```
{env:KEY:DEFAULTVALUE}
```

then the value will be retrieved as `os.environ['KEY']` and replace with DEFAULTVALUE if the environment variable does not exist.

If you specify a substitution string like this:

```
{env:KEY:}
```

then the value will be retrieved as `os.environ['KEY']` and replace with an empty string if the environment variable does not exist.

Substitutions can also be nested. In that case they are expanded starting from the innermost expression:

```
{env:KEY:{env:DEFAULT_OF_KEY}}
```

the above example is roughly equivalent to `os.environ.get('KEY', os.environ['DEFAULT_OF_KEY'])`

interactive shell substitution

It's possible to inject a config value only when tox is running in interactive shell (standard input):

> {tty:ON_VALUE:OFF_VALUE}

The first value is the value to inject when the interactive terminal is available, the second value is the value to use when it's not. The later on is optional. A good use case for this is e.g. passing in the `--pdb` flag for pytest.

substitutions for positional arguments in commands

New in version 1.0.

If you specify a substitution string like this:

```
{posargs:DEFAULTS}
```

then the value will be replaced with positional arguments as provided to the tox command:

```
tox arg1 arg2
```

In this instance, the positional argument portion will be replaced with `arg1 arg2`. If no positional arguments were specified, the value of DEFAULTS will be used instead. If DEFAULTS contains other substitution strings, such as `{env:*}`, they will be interpreted.,

Use a double `--` if you also want to pass options to an underlying test command, for example:

```
tox -- --opt1 ARG1
```

will make the `--opt1 ARG1` appear in all test commands where `[]` or `{posargs}` was specified. By default (see `args_are_paths` setting), `tox` rewrites each positional argument if it is a relative path and exists on the filesystem to become a path relative to the `changedir` setting.

Previous versions of tox supported the `[.*]` pattern to denote positional arguments with defaults. This format has been deprecated. Use `{posargs:DEFAULTS}` to specify those.

Substitution for values from other sections

New in version 1.4.

Values from other sections can be referred to via:

```
{[sectionname]valuename}
```

which you can use to avoid repetition of config values. You can put default values in one section and reference them in others to avoid repeating the same values:

```
[base]
deps =
    pytest
    mock
    pytest-xdist

[testenv:dulwich]
deps =
    dulwich
    {[base]deps}

[testenv:mercurial]
deps =
    mercurial
    {[base]deps}
```

5.3.4 Generating environments, conditional settings

New in version 1.8.

Suppose you want to test your package against python2.7, python3.6 and against several versions of a dependency, say Django 1.5 and Django 1.6. You can accomplish that by writing down 2*2 = 4 `[testenv:*]` sections and then listing all of them in `envlist`.

However, a better approach looks like this:

```
[tox]
envlist = {py27,py36}-django{15,16}

[testenv]
deps =
    pytest
    django15: Django>=1.5,<1.6
    django16: Django>=1.6,<1.7
    py36: unittest2
commands = pytest
```

This uses two new facilities of tox-1.8:

- generative envlist declarations where each envname consists of environment parts or "factors"
- "factor" specific settings

Let's go through this step by step.

Generative envlist

```
envlist = {py36,py27}-django{15,16}
```

This is bash-style syntax and will create `2*2=4` environment names like this:

```
py27-django15
py27-django16
py36-django15
py36-django16
```

You can still list environments explicitly along with generated ones:

```
envlist = {py27,py36}-django{15,16}, docs, flake
```

Keep in mind that whitespace characters (except newline) within `{}` are stripped, so the following line defines the same environment names:

```
envlist = {py27,py36}-django{ 15, 16 }, docs, flake
```

Note: To help with understanding how the variants will produce section values, you can ask tox to show their expansion with a new option:

```
$ tox -l
py27-django15
py27-django16
py36-django15
py36-django16
docs
flake
```

Factors and factor-conditional settings

Parts of an environment name delimited by hyphens are called factors and can be used to set values conditionally. In list settings such as `deps` or `commands` you can freely intermix optional lines with unconditional ones:

```
[testenv]
deps =
    pytest
    django15: Django>=1.5,<1.6
    django16: Django>=1.6,<1.7
    py36: unittest2
```

Reading it line by line:

- `pytest` will be included unconditionally,
- `Django>=1.5,<1.6` will be included for environments containing `django15` factor,

- `Django>=1.6,<1.7` similarly depends on `django16` factor,
- `unittest` will be loaded for Python 3.6 environments.

tox provides a number of default factors corresponding to Python interpreter versions. The conditional setting above will lead to either `python3.6` or `python2.7` used as base python, e.g. `python3.6` is selected if current environment contains `py36` factor.

Note: Configuring `basepython` for environments using default factors will result in a warning. Configure `ignore_basepython_conflict` if you wish to explicitly ignore these conflicts, allowing you to define a global `basepython` for all environments *except* those with default factors.

Complex factor conditions

Sometimes you need to specify the same line for several factors or create a special case for a combination of factors. Here is how you do it:

```
[tox]
envlist = py{27,34,36}-django{15,16}-{sqlite,mysql}

[testenv]
deps =
    py34-mysql: PyMySQL     ; use if both py34 and mysql are in the env name
    py27,py36: urllib3      ; use if either py36 or py27 are in the env name
    py{27,36}-sqlite: mock  ; mocking sqlite in python 2.x & 3.6
    !py34-sqlite: mock      ; mocking sqlite, except in python 3.4
    sqlite-!py34: mock      ; (same as the line above)
```

Take a look at the first `deps` line. It shows how you can special case something for a combination of factors, by just hyphenating the combining factors together. This particular line states that `PyMySQL` will be loaded for python 3.3, mysql environments, e.g. `py34-django15-mysql` and `py34-django16-mysql`.

The second line shows how you use the same setting for several factors - by listing them delimited by commas. It's possible to list not only simple factors, but also their combinations like `py27-sqlite,py36-sqlite`.

The remaining lines all have the same effect and use conditions equivalent to `py27-sqlite,py36-sqlite`. They have all been added only to help demonstrate the following:

- how factor expressions get expanded the same way as in envlist
- how to use negated factor conditions by prefixing negated factors with `!`
- that the order in which factors are hyphenated together does not matter

Note: Factors don't do substring matching against env name, instead every hyphenated expression is split by `-` and if ALL of its non-negated factors and NONE of its negated ones are also factors of an env then that condition is considered to hold for that env.

For example, environment `py36-mysql-!dev`:

- would be matched by expressions `py36`, `py36-mysql` or `mysql-py36`,
- but not `py2`, `py36-sql` or `py36-mysql-dev`.

5.3.5 Advanced settings

Handle interpreter directives with long lengths

For systems supporting executable text files (scripts with a shebang), the system will attempt to parse the interpreter directive to determine the program to execute on the target text file. When `tox` prepares a virtual environment in a file container which has a large length (e.x. using Jenkins Pipelines), the system might not be able to invoke shebang scripts which define interpreters beyond system limits (e.x. Linux as a limit of 128; `BINPRM_BUF_SIZE`). To workaround an environment which suffers from an interpreter directive limit, a user can bypass the system's interpreter parser by defining the `TOX_LIMITED_SHEBANG` environment variable before invoking `tox`:

```
export TOX_LIMITED_SHEBANG=1
```

When the workaround is enabled, all tox-invoked text file executables will have their interpreter directive parsed by and explicitly executed by `tox`.

5.4 Injected environment variables

tox will inject the following environment variables that you can use to test that your command is running within tox:

New in version 3.4.

- `TOX_WORK_DIR` env var is set to the tox work directory
- `TOX_ENV_NAME` is set to the current running tox environment name
- `TOX_ENV_DIR` is set to the current tox environments working dir.

  note this applies for all tox envs (isolated packaging too) and all external commands called (e.g. install command - pip).

5.5 Other Rules and notes

- `path` specifications: if a specified `path` is a relative path it will be considered as relative to the `toxinidir`, the directory where the configuration file resides.

5.6 support and contact channels

Getting in contact:

- join the tox-dev mailing list for tox related questions and development discussions.
- file a report on the issue tracker
- hang out on the irc.freenode.net #pylib channel
- fork the github repository and submit merge/pull requests (see the developers help page – *Developers FAQ*)

5.6.1 paid professional support

contact holger at merlinux.eu, an association of experienced well-known Python developers.

5.7 Changelog history

Versions follow Semantic Versioning (`<major>.<minor>.<patch>`).

Backward incompatible (breaking) changes will only be introduced in major versions with advance notice in the **Deprecations** section of releases.

5.7.1 3.4.0 (2018-09-20)

Bugfixes

- add `--exists-action w` to default pip flags to handle better VCS dependencies (pip documentation on this) - by @gaborbernat (#503)
- instead of assuming the Python version from the base python name ask the interpreter to reveal the version for the `ignore_basepython_conflict` flag - by @gaborbernat (#908)
- PEP-517 packaging fails with sdist already exists, fixed via ensuring the dist folder is empty before invoking the backend and pypa/setuptools 1481 - by @gaborbernat (#1003)

Features

- add `commands_pre` and `commands_post` that run before and after running the `commands` (setup runs always, commands only if setup suceeds, teardown always - all run until the first failing command) - by @gaborbernat (#167)
- `pyproject.toml` config support initially by just inline the tox.ini under `tool.tox.legacy_tox_ini` key; config source priority order is `pyproject.toml`, `tox.ini` and then `setup.cfg` - by @gaborbernat (#814)
- use the os environment variable `TOX_SKIP_ENV` to filter out tox environment names from the run list (set by `envlist`) - by @gaborbernat (#824)
- always set `PIP_USER=0` (do not install into the user site package, but inside the virtual environment created) and `PIP_NO_DEPS=0` (installing without dependencies can cause broken package installations) inside tox - by @gaborbernat (#838)
- tox will inject some environment variables that to indicate a command is running within tox: `TOX_WORK_DIR` env var is set to the tox work directory, `TOX_ENV_NAME` is set to the current running tox environment name, `TOX_ENV_DIR` is set to the current tox environments working dir - by @gaborbernat (#847)
- While running tox invokes various commands (such as building the package, pip installing dependencies and so on), these were printed in case they failed as Python arrays. Changed the representation to a shell command, allowing the users to quickly replicate/debug the failure on their own - by @gaborbernat (#851)
- skip missing interpreters value from the config file can now be overridden via the `--skip-missing-interpreters` cli flag - by @gaborbernat (#903)
- keep additional environments config order when listing them - by @gaborbernat (#921)
- allow injecting config value inside the ini file dependent of the fact that we're connected to an interactive shell or not - by @gaborbernat (#947)
- do not build sdist if skip install is specified for the envs to be run - by @gaborbernat (#974)
- when verbosity level increases above two start passing through verbosity flags to pip - by @gaborbernat (#982)
- when discovering the interpreter to use check if the tox host Python matches and use that if so - by @gaborbernat (#994)

- `-vv` will print out why a virtual environment is re-created whenever this operation is triggered - by @gaborbernat (#1004)

Documentation

- clarify that `python` and `pip` refer to the virtual environments executable - by @gaborbernat (#305)
- add Sphinx and mkdocs example of generating documentation via tox - by @gaborbernat (#374)
- specify that `setup.cfg` tox configuration needs to be inside the `tox:tox` namespace - by @gaborbernat (#545)

5.7.2 3.3.0 (2018-09-11)

Bugfixes

- fix `TOX_LIMITED_SHEBANG` when running under python3 - by @asottile (#931)

Features

- PEP-517 source distribution support (create a `.package` virtual environment to perform build operations inside) by @gaborbernat (#573)
- flit support via implementing `PEP-517` by @gaborbernat (#820)
- packaging now is exposed as a hook via `tox_package(session, venv)` - by @gaborbernat (#951)

Miscellaneous

- Updated the VSTS build YAML to use the latest jobs and pools syntax - by @davidstaheli (#955)

5.7.3 3.2.1 (2018-08-10)

Bugfixes

- `--parallel--safe-build` no longer cleans up its folders (`distdir`, `distshare`, `log`). - by @gaborbernat (#849)

5.7.4 3.2.0 (2018-08-10)

Features

- Switch pip invocations to use the module `-m pip` instead of direct invocation. This could help avoid some of the shebang limitations. - by @gaborbernat (#935)
- Ability to specify package requirements for the tox run via the `tox.ini` (`tox` section under key `requires` - PEP-508 style): can be used to specify both plugin requirements or build dependencies. - by @gaborbernat (#783)
- Allow to run multiple tox instances in parallel by providing the `--parallel--safe-build` flag. - by @gaborbernat (#849)

5.7.5 3.1.3 (2018-08-03)

Bugfixes

- A caching issue that caused the `develop-inst-nodeps` action, which reinstalls the package under test, to always run has been resolved. The `develop-inst-noop` action, which, as the name suggests, is a no-op, will now run unless there are changes to `setup.py` or `setup.cfg` files that have not been reflected - by @stephenfin (#909)

Features

- Python version testenvs are now automatically detected instead of comparing against a hard-coded list of supported versions. This enables `py38` and eventually `py39` / `py40` / etc. to work without requiring an upgrade to `tox`. As such, the following public constants are now deprecated (and scheduled for removal in `tox` 4.0: `CPYTHON_VERSION_TUPLES`, `PYPY_VERSION_TUPLES`, `OTHER_PYTHON_INTERPRETERS`, and `DEFAULT_FACTORS` - by @asottile (#914)

Documentation

- Add a system overview section on the index page that explains briefly how tox works - by @gaborbernat. (#867)

5.7.6 3.1.2 (2018-07-12)

Bugfixes

- Revert "Fix bug with incorrectly defactorized dependencies (#772)" due to a regression ((#799)) - by @obestwalter

5.7.7 3.1.1 (2018-07-09)

Bugfixes

- PyPi documentation for `3.1.0` is broken. Added test to check for this, and fix it by @gaborbernat. (#879)

5.7.8 3.1.0 (2018-07-08)

Bugfixes

- Add `ignore_basepython_conflict`, which determines whether conflicting `basepython` settings for environments containing default factors, such as `py27` or `django18-py35`, should be ignored or result in warnings. This was a common source of misconfiguration and is rarely, if ever, desirable from a user perspective - by @stephenfin (#477)
- Fix bug with incorrectly defactorized dependencies (deps passed to pip were not de-factorized) - by @bartsanchez (#706)

Features

- Add support for multiple PyPy versions using default factors. This allows you to use, for example, `pypy27` knowing that the correct intepreter will be used by default - by @stephenfin (#19)
- Add support to explicitly invoke interpreter directives for environments with long path lengths. In the event that `tox` cannot invoke scripts with a system-limited shebang (e.x. a Linux host running a Jenkins Pipeline), a user can set the environment variable `TOX_LIMITED_SHEBANG` to workaround the system's limitation (e.x. `export TOX_LIMITED_SHEBANG=1`) - by @jdknight (#794)
- introduce a constants module to be used internally and as experimental API - by @obestwalter (#798)
- Make `py2` and `py3` aliases also resolve via `py` on windows by @asottile. This enables the following things: `tox -e py2` and `tox -e py3` work on windows (they already work on posix); and setting `basepython=python2` or `basepython=python3` now works on windows. (#856)
- Replace the internal version parsing logic from the not well tested PEP-386 parser for the more general PEP-440. packaging >= 17.1 is now an install dependency by @gaborbernat. (#860)

Documentation

- extend the plugin documentation and make lot of small fixes and improvements - by @obestwalter (#797)
- tidy up tests - remove unused fixtures, update old cinstructs, etc. - by @obestwalter (#799)
- Various improvements to documentation: open browser once documentation generation is done, show Github/Travis info on documentation page, remove duplicate header for changelog, generate unreleased news as DRAFT on top of changelog, make the changelog page more compact and readable (width up to 1280px) by @gaborbernat (#859)

Miscellaneous

- filter out unwanted files in package - by @obestwalter (#754)
- make the already existing implicit API explicit - by @obestwalter (#800)
- improve tox quickstart and corresponding tests - by @obestwalter (#801)
- tweak codecov settings via .codecov.yml - by @obestwalter (#802)

5.7.9 3.0.0 (2018-04-02)

Bugfixes

- Write directly to stdout buffer if possible to prevent str vs bytes issues - by @asottile (#426)
- fix #672 reporting to json file when skip-missing-interpreters option is used - by @r2dan (#672)
- avoid `Requested Python version (X.Y) not installed` stderr output when a Python environment is looked up using the `py` Python launcher on Windows and the environment is not found installed on the system - by @jurko-gospodnetic (#692)
- Fixed an issue where invocation of tox from the Python package, where invocation errors (failed actions) occur results in a change in the sys.stdout stream encoding in Python 3.x. New behaviour is that sys.stdout is reset back to its original encoding after invocation errors - by @tonybaloney (#723)
- The reading of command output sometimes failed with `IOError: [Errno 0] Error` on Windows, this was fixed by using a simpler method to update the read buffers. - by @fschulze (#727)

- (only affected rc releases) fix up tox.cmdline to be callable without args - by @gaborbernat. (#773)
- (only affected rc releases) Revert breaking change of tox.cmdline not callable with no args - by @gaborbernat. (#773)
- (only affected rc releases) fix #755 by reverting the `cmdline` import to the old location and changing the entry point instead - by @fschulze (#755)

Features

- `tox` displays exit code together with `InvocationError` - by @blueyed and @ederag. (#290)
- Hint for possible signal upon `InvocationError`, on posix systems - by @ederag and @asottile. (#766)
- Add a `-q` option to progressively silence tox's output. For each time you specify `-q` to tox, the output provided by tox reduces. This option allows you to see only your command output without the default verbosity of what tox is doing. This also counter-acts usage of `-v`. For example, running `tox -v -q ...` will provide you with the default verbosity. `tox -vv -q` is equivalent to `tox -v`. By @sigmavirus24 (#256)
- add support for negated factor conditions, e.g. `!dev: production_log` - by @jurko-gospodnetic (#292)
- Headings like `installed: <packages>` will not be printed if there is no output to display after the :, unless verbosity is set. By @cryvate (#601)
- Allow spaces in command line options to pip in deps. Where previously only `deps=-rreq.txt` and `deps=--requirement=req.txt` worked, now also `deps=-r req.txt` and `deps=--requirement req.txt` work - by @cryvate (#668)
- drop Python `2.6` and `3.3` support: `setuptools` dropped supporting these, and as we depend on it we'll follow up with doing the same (use `tox <= 2.9.1` if you still need this support) - by @gaborbernat (#679)
- Add tox_runenvreport as a possible plugin, allowing the overriding of the default behaviour to execute a command to get the installed packages within a virtual environment - by @tonybaloney (#725)
- Forward `PROCESSOR_ARCHITECTURE` by default on Windows to fix `platform.machine()`. (#740)

Documentation

- Change favicon to the vector beach ball - by @hazalozturk (#748)
- Change sphinx theme to alabaster and add logo/favicon - by @hazalozturk (#639)

Miscellaneous

- Running `tox` without a `setup.py` now has a more friendly error message and gives troubleshooting suggestions - by @Volcyy. (#331)
- Fix pycodestyle (formerly pep8) errors E741 (ambiguous variable names, in this case, 'l's) and remove ignore of this error in tox.ini - by @cryvate (#663)
- touched up `interpreters.py` code and added some missing tests for it - by @jurko-gospodnetic (#708)
- The `PYTHONDONTWRITEBYTECODE` environment variable is no longer unset - by @stephenfin. (#744)

5.7.10 2.9.1 (2017-09-29)

Miscellaneous

- integrated new release process and fixed changelog rendering for pypi.org - by @obestwalter.

5.7.11 2.9.0 (2017-09-29)

Features

- `tox --version` now shows information about all registered plugins - by @obestwalter (#544)

Bugfixes

- `skip_install` overrides `usedevelop` (`usedevelop` is an option to choose the installation type if the package is installed and `skip_install` determines if it should be installed at all) - by @ferdonline (#571)

Miscellaneous

- #635 inherit from correct exception - by @obestwalter (#635).
- spelling and escape sequence fixes - by @scoop (#637 and #638).
- add a badge to show build status of documentation on readthedocs.io - by @obestwalter.

Documentation

- add towncrier to allow adding changelog entries with the pull requests without generating merge conflicts; with this release notes are now grouped into four distinct collections: `Features`, `Bugfixes`, `Improved Documentation` and `Deprecations and Removals`. (#614)

5.7.12 2.8.2 (2017-10-09)

- #466: stop env var leakage if popen failed with resultjson or redirect

5.7.13 2.8.1 (2017-09-04)

- pull request 599: fix problems with implementation of #515. Substitutions from other sections were not made anymore if they were not in `envlist`. Thanks to Clark Boylan (@cboylan) for helping to get this fixed (pull request 597).

5.7.14 2.8.0 (2017-09-01)

- #276: Remove easy_install from docs (TL;DR: use pip). Thanks Martin Andrysík (@sifuraz).
- #301: Expand nested substitutions in `tox.ini`. Thanks @vlaci. Thanks to Eli Collins (@eli-collins) for creating a reproducer.
- #315: add `--help` and `--version` to helptox-quickstart. Thanks @vlaci.
- #326: Fix `OSError` 'Not a directory' when creating env on Jython 2.7.0. Thanks Nick Douma (@LordGaav).

- #429: Forward `MSYSTEM` by default on Windows. Thanks Marius Gedminas (@mgedmin) for reporting this.
- #449: add multi platform example to the docs. Thanks Aleks Bunin (@sashkab) and @rndr.
- #474: Start using setuptools_scm for tag based versioning.
- #484: Renamed `py.test` to `pytest` throughout the project. Thanks Slam (@3lnc).
- #504: With `-a`: do not show additional environments header if there are none. Thanks @rndr.
- #515: Don't require environment variables in test environments where they are not used. Thanks André Caron (@AndreLouisCaron).
- #517: Forward `NUMBER_OF_PROCESSORS` by default on Windows to fix `multiprocessor.cpu_count()`. Thanks André Caron (@AndreLouisCaron).
- #518: Forward `USERPROFILE` by default on Windows. Thanks André Caron (@AndreLouisCaron).
- pull request 528: Fix some of the warnings displayed by pytest 3.1.0. Thanks Bruno Oliveira (@nicoddemus).
- pull request 547: Add regression test for #137. Thanks Martin Andrysík (@sifuraz).
- pull request 553: Add an XFAIL test to reproduce upstream bug #203. Thanks Bartolomé Sánchez Salado (@bartsanchez).
- pull request 556: Report more meaningful errors on why virtualenv creation failed. Thanks @vlaci. Also thanks to Igor Sadchenko (@igor-sadchenko) for pointing out a problem with that PR before it hit the masses
- pull request 575: Add announcement doc to end all announcement docs (using only `CHANGELOG` and Github issues since 2.5 already).
- pull request 580: Do not ignore Sphinx warnings anymore. Thanks Bernát Gábor (@gaborbernat).
- pull request 585: Expand documentation to explain pass through of flags from deps to pip (e.g. `-rrequirements.txt`, `-cconstraints.txt`). Thanks Alexander Loechel (@loechel).
- pull request 588: Run pytest wit xfail_strict and adapt affected tests.

5.7.15 2.7.0 (2017-04-02)

- pull request 450: Stop after the first installdeps and first testenv create hooks succeed. This changes the default behaviour of `tox_testenv_create` and `tox_testenv_install_deps` to not execute other registered hooks when the first hook returns a result that is not `None`. Thanks Anthony Sottile (@asottile).
- #271 and #464: Improve environment information for users.

  New command line parameter: `-a` show **all** defined environments - not just the ones defined in (or generated from) envlist.

  New verbosity settings for `-l` and `-a`: show user defined descriptions of the environments. This also works for generated environments from factors by concatenating factor descriptions into a complete description.

  Note that for backwards compatibility with scripts using the output of `-l` it's output remains unchanged.

  Thanks Bernát Gábor (@gaborbernat).
- #464: Fix incorrect egg-info location for modified package_dir in setup.py. Thanks Selim Belhaouane (@selimb).
- #431: Add 'LANGUAGE' to default passed environment variables. Thanks Paweł Adamczak (@pawelad).
- #455: Add a Vagrantfile with a customized Arch Linux box for local testing. Thanks Oliver Bestwalter (@obestwalter).
- #454: Revert pull request 407, empty commands is not treated as an error. Thanks Anthony Sottile (@asottile).

- #446: (infrastructure) Travis CI tests for tox now also run on OS X now. Thanks Jason R. Coombs (@jaraco).

5.7.16 2.6.0 (2017-02-04)

- add "alwayscopy" config option to instruct virtualenv to always copy files instead of symlinking. Thanks Igor Duarte Cardoso (@igordcard).
- pass setenv variables to setup.py during a usedevelop install. Thanks Eli Collins (@eli-collins).
- replace all references to testrun.org with readthedocs ones. Thanks Oliver Bestwalter (@obestwalter).
- fix #323 by avoiding virtualenv14 is not used on py32 (although we don't officially support py32). Thanks Jason R. Coombs (@jaraco).
- add Python 3.6 to envlist and CI. Thanks Andrii Soldatenko (@andriisoldatenko).
- fix glob resolution from TOX_TESTENV_PASSENV env variable Thanks Allan Feldman (@a-feld).

5.7.17 2.5.0 (2016-11-16)

- slightly backward incompatible: fix #310: the {posargs} substitution now properly preserves the tox command line positional arguments. Positional arguments with spaces are now properly handled. NOTE: if your tox invocation previously used extra quoting for positional arguments to work around #310, you need to remove the quoting. Example: tox – "'some string'" # has to now be written simply as tox – "some string" thanks holger krekel. You can set `minversion = 2.5.0` in the `[tox]` section of `tox.ini` to make sure people using your tox.ini use the correct version.
- fix #359: add COMSPEC to default passenv on windows. Thanks @anthrotype.
- add support for py36 and py37 and add py36-dev and py37(nightly) to travis builds of tox. Thanks John Vandenberg.
- fix #348: add py2 and py3 as default environments pointing to "python2" and "python3" basepython executables. Also fix #347 by updating the list of default envs in the tox basic example. Thanks Tobias McNulty.
- make "-h" and "–help-ini" options work even if there is no tox.ini, thanks holger krekel.
- add {:} substitution, which is replaced with os-specific path separator, thanks Lukasz Rogalski.
- fix #305: `downloadcache` test env config is now ignored as pip-8 does caching by default. Thanks holger krekel.
- output from install command in verbose (-vv) mode is now printed to console instead of being redirected to file, thanks Lukasz Rogalski
- fix #399. Make sure {envtmpdir} is created if it doesn't exist at the start of a testenvironment run. Thanks Manuel Jacob.
- fix #316: Lack of commands key in ini file is now treated as an error. Reported virtualenv status is 'nothing to do' instead of 'commands succeeded', with relevant error message displayed. Thanks Lukasz Rogalski.

5.7.18 2.4.1 (2016-10-12)

- fix #380: properly perform substitution again. Thanks Ian Cordasco.

5.7.19 2.4.0 (2016-10-12)

- remove PYTHONPATH from environment during the install phase because a tox-run should not have hidden dependencies and the test commands will also not see a PYTHONPATH. If this causes unforeseen problems it may be reverted in a bugfix release. Thanks Jason R. Coombs.
- fix #352: prevent a configuration where envdir==toxinidir and refine docs to warn people about changing "envdir". Thanks Oliver Bestwalter and holger krekel.
- fix #375, fix #330: warn against tox-setup.py integration as "setup.py test" should really just test with the current interpreter. Thanks Ronny Pfannschmidt.
- fix #302: allow cross-testenv substitution where we substitute with `{x,y}` generative syntax. Thanks Andrew Pashkin.
- fix #212: allow escaping curly brace chars "{" and "}" if you need the chars "{" and "}" to appear in your commands or other ini values. Thanks John Vandenberg.
- addresses #66: add –workdir option to override where tox stores its ".tox" directory and all of the virtualenv environment. Thanks Danring.
- introduce per-venv list_dependencies_command which defaults to "pip freeze" to obtain the list of installed packages. Thanks Ted Shaw, Holger Krekel.
- close #66: add documentation to jenkins page on how to avoid "too long shebang" lines when calling pip from tox. Note that we can not use "python -m pip install X" by default because the latter adds the CWD and pip will think X is installed if it is there. "pip install X" does not do that.
- new list_dependencies_command to influence how tox determines which dependencies are installed in a testenv.
- (experimental) New feature: When a search for a config file fails, tox tries loading setup.cfg with a section prefix of "tox".
- fix #275: Introduce hooks ``tox_runtest_pre` `` and `tox_runtest_post` which run before and after the tests of a venv, respectively. Thanks to Matthew Schinckel and itxaka serrano.
- fix #317: evaluate minversion before tox config is parsed completely. Thanks Sachi King for the PR.
- added the "extras" environment option to specify the extras to use when doing the sdist or develop install. Contributed by Alex Grönholm.
- use pytest-catchlog instead of pytest-capturelog (latter is not maintained, uses deprecated pytest API)

5.7.20 2.3.2 (2016-02-11)

- fix #314: fix command invocation with .py scripts on windows.
- fix #279: allow cross-section substitution when the value contains posargs. Thanks Sachi King for the PR.

5.7.21 2.3.1 (2015-12-14)

- fix #294: re-allow cross-section substitution for setenv.

5.7.22 2.3.0 (2015-12-09)

- DEPRECATE use of "indexservers" in tox.ini. It complicates the internal code and it is recommended to rather use the devpi system for managing indexes for pip.

- fix #285: make setenv processing fully lazy to fix regressions of tox-2.2.X and so that we can now have testenv attributes like "basepython" depend on environment variables that are set in a setenv section. Thanks Nelfin for some tests and initial work on a PR.
- allow "#" in commands. This is slightly incompatible with commands sections that used a comment after a "" line continuation. Thanks David Stanek for the PR.
- fix #289: fix build_sphinx target, thanks Barry Warsaw.
- fix #252: allow environment names with special characters. Thanks Julien Castets for initial PR and patience.
- introduce experimental tox_testenv_create(venv, action) and tox_testenv_install_deps(venv, action) hooks to allow plugins to do additional work on creation or installing deps. These hooks are experimental mainly because of the involved "venv" and session objects whose current public API is not fully guaranteed.
- internal: push some optional object creation into tests because tox core doesn't need it.

5.7.23 2.2.1 (2015-12-09)

- fix bug where {envdir} substitution could not be used in setenv if that env value is then used in {basepython}. Thanks Florian Bruhin.

5.7.24 2.2.0 (2015-11-11)

- fix #265 and add LD_LIBRARY_PATH to passenv on linux by default because otherwise the python interpreter might not start up in certain configurations (redhat software collections). Thanks David Riddle.
- fix #246: fix regression in config parsing by reordering such that {envbindir} can be used again in tox.ini. Thanks Olli Walsh.
- fix #99: the {env:. . . } substitution now properly uses environment settings from the `setenv` section. Thanks Itxaka Serrano.
- fix #281: make –force-dep work when urls are present in dependency configs. Thanks Glyph Lefkowitz for reporting.
- fix #174: add new `ignore_outcome` testenv attribute which can be set to True in which case it will produce a warning instead of an error on a failed testenv command outcome. Thanks Rebecka Gulliksson for the PR.
- fix #280: properly skip missing interpreter if {envsitepackagesdir} is present in commands. Thanks BB:ceridwenv

5.7.25 2.1.1 (2015-06-23)

- fix platform skipping for detox
- report skipped platforms as skips in the summary

5.7.26 2.1.0 (2015-06-19)

- fix #258, fix #248, fix #253: for non-test commands (installation, venv creation) we pass in the full invocation environment.
- remove experimental –set-home option which was hardly used and hackily implemented (if people want home-directory isolation we should figure out a better way to do it, possibly through a plugin)
- fix #259: passenv is now a line-list which allows to intersperse comments. Thanks stefano-m.

- allow envlist to be a multi-line list, to intersperse comments and have long envlist settings split more naturally. Thanks Andre Caron.
- introduce a TOX_TESTENV_PASSENV setting which is honored when constructing the set of environment variables for test environments. Thanks Marc Abramowitz for pushing in this direction.

5.7.27 2.0.2 (2015-06-03)

- fix #247: tox now passes the LANG variable from the tox invocation environment to the test environment by default.
- add SYSTEMDRIVE into default passenv on windows to allow pip6 to work. Thanks Michael Krause.

5.7.28 2.0.1 (2015-05-13)

- fix wheel packaging to properly require argparse on py26.

5.7.29 2.0.0 (2015-05-12)

- (new) introduce environment variable isolation: tox now only passes the PATH and PIP_INDEX_URL variable from the tox invocation environment to the test environment and on Windows also `SYSTEMROOT`, `PATHEXT`, `TEMP` and `TMP` whereas on unix additionally `TMPDIR` is passed. If you need to pass through further environment variables you can use the new `passenv` setting, a space-separated list of environment variable names. Each name can make use of fnmatch-style glob patterns. All environment variables which exist in the tox-invocation environment will be copied to the test environment.
- a new `--help-ini` option shows all possible testenv settings and their defaults.
- (new) introduce a way to specify on which platform a testenvironment is to execute: the new per-venv "platform" setting allows to specify a regular expression which is matched against sys.platform. If platform is set and doesn't match the platform spec in the test environment the test environment is ignored, no setup or tests are attempted.
- **(new) add per-venv "ignore_errors" setting, which defaults to False.** If `True`, a non-zero exit code from one command will be ignored and further commands will be executed (which was the default behavior in tox < 2.0). If `False` (the default), then a non-zero exit code from one command will abort execution of commands for that environment.
- show and store in json the version dependency information for each venv
- remove the long-deprecated "distribute" option as it has no effect these days.
- fix #233: avoid hanging with tox-setuptools integration example. Thanks simonb.
- fix #120: allow substitution for the commands section. Thanks Volodymyr Vitvitski.
- fix #235: fix AttributeError with –installpkg. Thanks Volodymyr Vitvitski.
- tox has now somewhat pep8 clean code, thanks to Volodymyr Vitvitski.
- fix #240: allow to specify empty argument list without it being rewritten to ".". Thanks Daniel Hahler.
- introduce experimental (not much documented yet) plugin system based on pytest's externalized "pluggy" system. See tox/hookspecs.py for the current hooks.
- introduce parser.add_testenv_attribute() to register an ini-variable for testenv sections. Can be used from plugins through the tox_add_option hook.

- rename internal files – tox offers no external API except for the experimental plugin hooks, use tox internals at your own risk.
- DEPRECATE distshare in documentation

5.7.30 1.9.2 (2015-03-23)

- backout ability that –force-dep substitutes name/versions in requirement files due to various issues. This fixes #228, fixes #230, fixes #231 which popped up with 1.9.1.

5.7.31 1.9.1 (2015-03-23)

- use a file instead of a pipe for command output in "–result-json". Fixes some termination issues with python2.6.
- allow –force-dep to override dependencies in "-r" requirements files. Thanks Sontek for the PR.
- fix #227: use "-m virtualenv" instead of "-mvirtualenv" to make it work with pyrun. Thanks Marc-Andre Lemburg.

5.7.32 1.9.0 (2015-02-24)

- fix #193: Remove `--pre` from the default `install_command`; by default tox will now only install final releases from PyPI for unpinned dependencies. Use `pip_pre = true` in a testenv or the `--pre` command-line option to restore the previous behavior.
- fix #199: fill resultlog structure ahead of virtualenv creation
- refine determination if we run from Jenkins, thanks Borge Lanes.
- echo output to stdout when `--report-json` is used
- fix #11: add a `skip_install` per-testenv setting which prevents the installation of a package. Thanks Julian Krause.
- fix #124: ignore command exit codes; when a command has a "-" prefix, tox will ignore the exit code of that command
- fix #198: fix broken envlist settings, e.g. {py26,py27}{-lint,}
- fix #191: lessen factor-use checks

5.7.33 1.8.1 (2014-10-24)

- fix #190: allow setenv to be empty.
- allow escaping curly braces with "". Thanks Marc Abramowitz for the PR.
- allow "." names in environment names such that "py27-django1.7" is a valid environment name. Thanks Alex Gaynor and Alex Schepanovski.
- report subprocess exit code when execution fails. Thanks Marius Gedminas.

5.7.34 1.8.0 (2014-09-24)

- new multi-dimensional configuration support. Many thanks to Alexander Schepanovski for the complete PR with docs. And to Mike Bayer and others for testing and feedback.
- fix #148: remove "__PYVENV_LAUNCHER__" from os.environ when starting subprocesses. Thanks Steven Myint.
- fix #152: set VIRTUAL_ENV when running test commands, thanks Florian Ludwig.
- better report if we can't get version_info from an interpreter executable. Thanks Floris Bruynooghe.

5.7.35 1.7.2 (2014-07-15)

- fix #150: parse {posargs} more like we used to do it pre 1.7.0. The 1.7.0 behaviour broke a lot of OpenStack projects. See PR85 and the issue discussions for (far) more details, hopefully resulting in a more refined behaviour in the 1.8 series. And thanks to Clark Boylan for the PR.
- fix #59: add a config variable `skip-missing-interpreters` as well as command line option `--skip-missing-interpreters` which won't fail the build if Python interpreters listed in tox.ini are missing. Thanks Alexandre Conrad for PR104.
- fix #164: better traceback info in case of failing test commands. Thanks Marc Abramowitz for PR92.
- support optional env variable substitution, thanks Morgan Fainberg for PR86.
- limit python hashseed to 1024 on Windows to prevent possible memory errors. Thanks March Schlaich for the PR90.

5.7.36 1.7.1 (2014-03-28)

- fix #162: don't list python 2.5 as compatible/supported
- fix #158 and fix #155: windows/virtualenv properly works now: call virtualenv through "python -m virtualenv" with the same interpreter which invoked tox. Thanks Chris Withers, Ionel Maries Cristian.

5.7.37 1.7.0 (2014-01-29)

- don't lookup "pip-script" anymore but rather just "pip" on windows as this is a pip implementation detail and changed with pip-1.5. It might mean that tox-1.7 is not able to install a different pip version into a virtualenv anymore.
- drop Python2.5 compatibility because it became too hard due to the setuptools-2.0 dropping support. tox now has no support for creating python2.5 based environments anymore and all internal special-handling has been removed.
- merged PR81: new option –force-dep which allows to override tox.ini specified dependencies in setuptools-style. For example "–force-dep 'django<1.6'" will make sure that any environment using "django" as a dependency will get the latest 1.5 release. Thanks Bruno Oliveria for the complete PR.
- merged PR125: tox now sets "PYTHONHASHSEED" to a random value and offers a "–hashseed" option to repeat a test run with a specific seed. You can also use –hashsheed=noset to instruct tox to leave the value alone. Thanks Chris Jerdonek for all the work behind this.
- fix #132: removing zip_safe setting (so it defaults to false) to allow installation of tox via easy_install/eggs. Thanks Jenisys.

- fix #126: depend on virtualenv>=1.11.2 so that we can rely (hopefully) on a pip version which supports –pre. (tox by default uses to –pre). also merged in PR84 so that we now call "virtualenv" directly instead of looking up interpreters. Thanks Ionel Maries Cristian. This also fixes #140.
- fix #130: you can now set install_command=easy_install {opts} {packages} and expect it to work for repeated tox runs (previously it only worked when always recreating). Thanks jenisys for precise reporting.
- fix #129: tox now uses Popen(. . . , universal_newlines=True) to force creation of unicode stdout/stderr streams. fixes a problem on specific platform configs when creating virtualenvs with Python3.3. Thanks Jorgen Schäfer or investigation and solution sketch.
- fix #128: enable full substitution in install_command, thanks for the PR to Ronald Evers
- rework and simplify "commands" parsing and in particular posargs substitutions to avoid various win32/posix related quoting issues.
- make sure that the –installpkg option trumps any usedevelop settings in tox.ini or
- introduce –no-network to tox's own test suite to skip tests requiring networks
- introduce –sitepackages to force sitepackages=True in all environments.
- fix #105 – don't depend on an existing HOME directory from tox tests.

5.7.38 1.6.1 (2013-09-04)

- fix #119: {envsitepackagesdir} is now correctly computed and has a better test to prevent regression.
- fix #116: make 1.6 introduced behaviour of changing to a per-env HOME directory during install activities dependent on "–set-home" for now. Should re-establish the old behaviour when no option is given.
- fix #118: correctly have two tests use realpath(). Thanks Barry Warsaw.
- fix test runs on environments without a home directory (in this case we use toxinidir as the homedir)
- fix #117: python2.5 fix: don't use `--insecure` option because its very existence depends on presence of "ssl". If you want to support python2.5/pip1.3.1 based test environments you need to install ssl and/or use PIP_INSECURE=1 through `setenv`. section.
- fix #102: change to {toxinidir} when installing dependencies. this allows to use relative path like in "-rrequirements.txt".

5.7.39 1.6.0 (2013-08-15)

- fix #35: add new EXPERIMENTAL "install_command" testenv-option to configure the installation command with options for dep/pkg install. Thanks Carl Meyer for the PR and docs.
- fix #91: python2.5 support by vendoring the virtualenv-1.9.1 script and forcing pip<1.4. Also the default [py25] environment modifies the default installer_command (new config option) to use pip without the "–pre" option which was introduced with pip-1.4 and is now required if you want to install non-stable releases. (tox defaults to install with "–pre" everywhere).
- during installation of dependencies HOME is now set to a pseudo location ({envtmpdir}/pseudo-home). If an index url was specified a .pydistutils.cfg file will be written with an index_url setting so that packages defining `setup_requires` dependencies will not silently use your HOME-directory settings or PyPi.
- fix #1: empty setup files are properly detected, thanks Anthon van der Neuth
- remove toxbootstrap.py for now because it is broken.

- fix #109 and fix #111: multiple "-e" options are now combined (previously the last one would win). Thanks Anthon van der Neut.
- add –result-json option to write out detailed per-venv information into a json report file to be used by upstream tools.
- add new config options `usedevelop` and `skipsdist` as well as a command line option `--develop` to install the package-under-test in develop mode. thanks Monty Tailor for the PR.
- always unset PYTHONDONTWRITEBYTE because newer setuptools doesn't like it
- if a HOMEDIR cannot be determined, use the toxinidir.
- refactor interpreter information detection to live in new tox/interpreters.py file, tests in tests/test_interpreters.py.

5.7.40 1.5.0 (2013-06-22)

- fix #104: use setuptools by default, instead of distribute, now that setuptools has distribute merged.
- make sure test commands are searched first in the virtualenv
- re-fix #2 - add whitelist_externals to be used in `[testenv*]` sections, allowing to avoid warnings for commands such as `make`, used from the commands value.
- fix #97 - allow substitutions to reference from other sections (thanks Krisztian Fekete)
- fix #92 - fix {envsitepackagesdir} to actually work again
- show (test) command that is being executed, thanks Lukasz Balcerzak
- re-license tox to MIT license
- depend on virtualenv-1.9.1
- rename README.txt to README.rst to make bitbucket happier

5.7.41 1.4.3 (2013-02-28)

- use pip-script.py instead of pip.exe on win32 to avoid the lock exe file on execution issue (thanks Philip Thiem)
- introduce -l|–listenv option to list configured environments (thanks Lukasz Balcerzak)
- fix downloadcache determination to work according to docs: Only make pip use a download cache if PIP_DOWNLOAD_CACHE or a downloadcache=PATH testenv setting is present. (The ENV setting takes precedence)
- fix #84 - pypy on windows creates a bin not a scripts venv directory (thanks Lukasz Balcerzak)
- experimentally introduce –installpkg=PATH option to install a package rather than create/install an sdist package. This will still require and use tox.ini and tests from the current working dir (and not from the remote package).
- substitute {envsitepackagesdir} with the package installation directory (closes #72) (thanks g2p)
- issue #70 remove PYTHONDONTWRITEBYTECODE workaround now that virtualenv behaves properly (thanks g2p)
- merged tox-quickstart command, contributed by Marc Abramowitz, which generates a default tox.ini after asking a few questions
- fix #48 - win32 detection of pypy and other interpreters that are on PATH (thanks Gustavo Picon)

- fix grouping of index servers, it is now done by name instead of indexserver url, allowing to use it to separate dependencies into groups even if using the same default indexserver.
- look for "tox.ini" files in parent dirs of current dir (closes #34)
- the "py" environment now by default uses the current interpreter (sys.executable) make tox' own setup.py test execute tests with it (closes #46)
- change tests to not rely on os.path.expanduser (closes #60), also make mock session return args[1:] for more precise checking (closes #61) thanks to Barry Warsaw for both.

5.7.42 1.4.2 (2012-07-20)

- fix some tests which fail if /tmp is a symlink to some other place
- "python setup.py test" now runs tox tests via tox :) also added an example on how to do it for your project.

5.7.43 1.4.1 (2012-07-03)

- fix #41 better quoting on windows - you can now use "<" and ">" in deps specifications, thanks Chris Withers for reporting

5.7.44 1.4 (2012-06-13)

- fix #26 - no warnings on absolute or relative specified paths for commands
- fix #33 - commentchars are ignored in key-value settings allowing for specifying commands like: python -c "import sys ; print sys" which would formerly raise irritating errors because the ";" was considered a comment
- tweak and improve reporting
- refactor reporting and virtualenv manipulation to be more accessible from 3rd party tools
- support value substitution from other sections with the {[section]key} syntax
- fix #29 - correctly point to pytest explanation for importing modules fully qualified
- fix #32 - use –system-site-packages and don't pass –no-site-packages
- add python3.3 to the default env list, so early adopters can test
- drop python2.4 support (you can still have your tests run on
- fix the links/checkout howtos in the docs python-2.4, just tox itself requires 2.5 or higher.

5.7.45 1.3 2011-12-21

- fix: allow to specify wildcard filesystem paths when specifying dependencies such that tox searches for the highest version
- fix issue #21: clear PIP_REQUIRES_VIRTUALENV which avoids pip installing to the wrong environment, thanks to bb's streeter
- make the install step honour a testenv's setenv setting (thanks Ralf Schmitt)

5.7.46 1.2 2011-11-10

- remove the virtualenv.py that was distributed with tox and depend on >=virtualenv-1.6.4 (possible now since the latter fixes a few bugs that the inlining tried to work around)
- fix #10: work around UnicodeDecodeError when invoking pip (thanks Marc Abramowitz)
- fix a problem with parsing {posargs} in tox commands (spotted by goodwill)
- fix the warning check for commands to be installed in testenvironment (thanks Michael Foord for reporting)

5.7.47 1.1 (2011-07-08)

- fix #5 - don't require argparse for python versions that have it
- fix #6 - recreate virtualenv if installing dependencies failed
- fix #3 - fix example on frontpage
- fix #2 - warn if a test command does not come from the test environment
- fixed/enhanced: except for initial install always call "-U –no-deps" for installing the sdist package to ensure that a package gets upgraded even if its version number did not change. (reported on TIP mailing list and IRC)
- inline virtualenv.py (1.6.1) script to avoid a number of issues, particularly failing to install python3 environments from a python2 virtualenv installation.
- rework and enhance docs for display on readthedocs.org

5.7.48 1.0

- move repository and toxbootstrap links to https://bitbucket.org/hpk42/tox
- fix #7: introduce a "minversion" directive such that tox bails out if it does not have the correct version.
- fix #24: introduce a way to set environment variables for for test commands (thanks Chris Rose)
- fix #22: require virtualenv-1.6.1, obsoleting virtualenv5 (thanks Jannis Leidel) and making things work with pypy-1.5 and python3 more seamlessly
- toxbootstrap.py (used by jenkins build slaves) now follows the latest release of virtualenv
- fix #20: document format of URLs for specifying dependencies
- fix #19: substitute Hudson for Jenkins everywhere following the renaming of the project. NOTE: if you used the special [tox:hudson] section it will now need to be named [tox:jenkins].
- fix issue 23 / apply some ReST fixes
- change the positional argument specifier to use {posargs:} syntax and fix issues #15 and #10 by refining the argument parsing method (Chris Rose)
- remove use of inipkg lazy importing logic - the namespace/imports are anyway very small with tox.
- fix a fspath related assertion to work with debian installs which uses symlinks
- show path of the underlying virtualenv invocation and bootstrap virtualenv.py into a working subdir
- added a CONTRIBUTORS file

5.7.49 0.9

- fix pip-installation mixups by always unsetting PIP_RESPECT_VIRTUALENV (thanks Armin Ronacher)
- #1: Add a toxbootstrap.py script for tox, thanks to Sridhar Ratnakumar
- added support for working with different and multiple PyPI indexservers.
- new option: -r|–recreate to force recreation of virtualenv
- depend on py>=1.4.0 which does not contain or install the py.test anymore which is now a separate distribution "pytest".
- show logfile content if there is an error (makes CI output more readable)

5.7.50 0.8

- work around a virtualenv limitation which crashes if PYTHONDONTWRITEBYTECODE is set.
- run pip/easy installs from the environment log directory, avoids naming clashes between env names and dependencies (thanks ronny)
- require a more recent version of py lib
- refactor and refine config detection to work from a single file and to detect the case where a python installation overwrote an old one and resulted in a new executable. This invalidates the existing virtualenvironment now.
- change all internal source to strip trailing whitespaces

5.7.51 0.7

- use virtualenv5 (my own fork of virtualenv3) for now to create python3 environments, fixes a couple of issues and makes tox more likely to work with Python3 (on non-windows environments)
- add `sitepackages` option for testenv sections so that environments can be created with access to globals (default is not to have access, i.e. create environments with `--no-site-packages`.
- addressing #4: always prepend venv-path to PATH variable when calling subprocesses
- fix #2: exit with proper non-zero return code if there were errors or test failures.
- added unittest2 examples contributed by Michael Foord
- only allow 'True' or 'False' for boolean config values (lowercase / uppercase is irrelevant)
- recreate virtualenv on changed configurations

5.7.52 0.6

- fix OSX related bugs that could cause the caller's environment to get screwed (sorry). tox was using the same file as virtualenv for tracking the Python executable dependency and there also was confusion wrt links. this should be fixed now.
- fix long description, thanks Michael Foord

5.7.53 0.5

- initial release

5.8 tox plugins

New in version 2.0.

A growing number of hooks make tox modifiable in different phases of execution by writing plugins.

tox - like pytest and devpi - uses pluggy to provide an extension mechanism for pip-installable internal or devpi/PyPi-published plugins.

5.8.1 Using plugins

To start using a plugin you need to install it in the same environment where the tox host is installed.

e.g.:

```
$ pip install tox-travis
```

You can search for available plugins on PyPi by typing `pip search tox` and filter for packages that are prefixed `tox-` or contain the "plugin" in the description. You will get some output similar to this:

```
tox-pipenv (1.4.1)                  - A pipenv plugin for tox
tox-pyenv (1.1.0)                   - tox plugin that makes tox use ``pyenv which``␣
↪to find
                                      python executables
tox-globinterpreter (0.3)           - tox plugin to allow specification of␣
↪interpreter
                                      locationspaths to use
tox-venv (0.2.0)                    - Use python3 venvs for python3 tox testenvs
tox-cmake (0.1.1)                   - Build CMake projects using tox
tox-travis (0.10)                   - Seamless integration of tox into Travis CI
tox-py-backwards (0.1)              - tox plugin for py-backwards
tox-pytest-summary (0.1.2)          - tox + Py.test summary
tox-envreport (0.2.0)               - A tox-plugin to document the setup of used␣
↪virtual
                                      environments.
tox-no-internet (0.1.0)             - Workarounds for using tox with no internet␣
↪connection
tox-virtualenv-no-download (1.0.2)  - Disable virtualenv's download-by-default in tox
tox-run-command (0.4)               - tox plugin to run arbitrary commands in a␣
↪virtualenv
tox-pip-extensions (1.2.1)          - Augment tox with different installation␣
↪methods via
                                      progressive enhancement.
tox-run-before (0.1)                - tox plugin to run shell commands before the␣
↪test
                                      environments are created.
tox-docker (1.0.0)                  - Launch a docker instance around test runs
tox-bitbucket-status (1.0)          - Update bitbucket status for each env
tox-pipenv-install (1.0.3)          - Install packages from Pipfile
```

There might also be some plugins not (yet) available from PyPi that could be installed directly fom source hosters like Github or Bitbucket (or from a local clone). See the

To see what is installed you can call `tox --version` to get the version of the host and names and locations of all installed plugins:

```
3.0.0 imported from /home/ob/.virtualenvs/tmp/lib/python3.6/site-packages/tox/__init__
↪.py
registered plugins:
    tox-travis-0.10 at /home/ob/.virtualenvs/tmp/lib/python3.6/site-packages/tox_
↪travis/hooks.py
    detox-0.12 at /home/ob/.virtualenvs/tmp/lib/python3.6/site-packages/detox/tox_
↪proclimit.py
```

5.8.2 Creating a plugin

Start from a template

You can create a new tox plugin with all the bells and whistles via a Cookiecutter template (see cookiecutter-tox-plugin - this will create a complete pypi-releasable, documented project with license, documentation and CI.

```
$ pip install -U cookiecutter
$ cookiecutter gh:tox-dev/cookiecutter-tox-plugin
```

5.8.3 Tutorial: a minimal tox plugin

Note: This is the minimal implementation to demonstrate what is absolutely necessary to have a working plugin for internal use. To move from something like this to a publishable plugin you could apply `cookiecutter -f cookiecutter-tox-plugin` and adapt the code to the package based structure used in the cookiecutter.

Let us consider you want to extend tox behaviour by displaying fireworks at the end of a successful tox run (we won't go into the details of how to display fireworks though).

To create a working plugin you need at least a python project with a tox entry point and a python module implementing one or more of the pluggy based hooks tox specifies (using the `@tox.hookimpl` decorator as marker).

minimal structure:

```
$ mkdir tox-fireworks
$ cd tox-fireworks
$ touch tox_fireworks.py
$ touch setup.py
```

contents of `tox_fireworks.py`:

```
import pluggy

hookimpl = pluggy.HookimplMarker("tox")


@hookimpl
def tox_addoption(parser):
    """Add command line option to display fireworks on request."""


@hookimpl
def tox_configure(config):
    """Post process config after parsing."""
```

(continues on next page)

(continued from previous page)

```python
@hookimpl
def tox_runenvreport(config):
    """Display fireworks if all was fine and requested."""
```

Note: See *Hook specifications and related API* for details

contents of `setup.py`:

```python
from setuptools import setup

setup(
    name="tox-fireworks",
    py_modules=["tox_fireworks"],
    entry_points={"tox": ["fireworks = tox_fireworks"]},
    classifiers=["Framework:: tox"],
)
```

Using the **tox-** prefix in `tox-fireworks` is an established convention to be able to see from the project name that this is a plugin for tox. It also makes it easier to find with e.g. `pip search 'tox-'` once it is released on PyPi.

To make your new plugin discoverable by tox, you need to install it. During development you should install it with `-e` or `--editable`, so that changes to the code are immediately active:

```
$ pip install -e </path/to/tox-fireworks>
```

5.8.4 Publish your plugin to PyPi

If you think the rest of the world could profit using your plugin you can publish it to PyPi.

You need to add some more meta data to `setup.py` (see cookiecutter-tox-plugin for a complete example or consult the setup.py docs).

Note: Make sure your plugin project name is prefixed by `tox-` to be easy to find via e.g. `pip search tox-`

You can and publish it like:

```
$ cd </path/to/tox-fireworks>
$ python setup.py sdist bdist_wheel upload
```

Note: You could also use twine for secure uploads.

For more information about packaging and deploying Python projects see the Python Packaging Guide.

5.8.5 Hook specifications and related API

Hook specifications for tox - see https://pluggy.readthedocs.io/

`tox.hookspecs.`**`tox_addoption`**`(`*parser*`)`
add command line options to the argparse-style parser object.

`tox.hookspecs.`**`tox_configure`**`(`*config*`)`

Called after command line options are parsed and ini-file has been read.

Please be aware that the config object layout may change between major tox versions.

`tox.hookspecs.`**`tox_get_python_executable`**`(`*envconfig*`)`

Return a python executable for the given python base name.

The first plugin/hook which returns an executable path will determine it.

`envconfig` is the testenv configuration which contains per-testenv configuration, notably the `.envname` and `.basepython` setting.

`tox.hookspecs.`**`tox_package`**`(`*session*, *venv*`)`

Return the package to be installed for the given venv.

Called once for every environment.

`tox.hookspecs.`**`tox_runenvreport`**`(`*venv*, *action*`)`

Get the installed packages and versions in this venv.

This could be used for alternative (ie non-pip) package managers, this plugin should return a `list` of type `str`

`tox.hookspecs.`**`tox_runtest`**`(`*venv*, *redirect*`)`

Run the tests for this venv.

Note: This hook uses `firstresult=True` (see pluggy first result only) – hooks implementing this will be run until one returns non-`None`.

`tox.hookspecs.`**`tox_runtest_post`**`(`*venv*`)`

Perform arbitrary action after running tests for this venv.

This could be used to have per-venv test reporting of pass/fail status.

`tox.hookspecs.`**`tox_runtest_pre`**`(`*venv*`)`

Perform arbitrary action before running tests for this venv.

This could be used to indicate that tests for a given venv have started, for instance.

`tox.hookspecs.`**`tox_testenv_create`**`(`*venv*, *action*`)`

Perform creation action for this venv.

Some example usage:

- To *add* behavior but still use tox's implementation to set up a virtualenv, implement this hook but do not return a value (or explicitly return `None`).
- To *override* tox's virtualenv creation, implement this hook and return a non-`None` value.

Note: This api is experimental due to the unstable api of *`tox.venv.VirtualEnv`*.

Note: This hook uses `firstresult=True` (see pluggy first result only) – hooks implementing this will be run until one returns non-`None`.

`tox.hookspecs.`**`tox_testenv_install_deps`**`(`*venv*, *action*`)`

Perform install dependencies action for this venv.

Some example usage:

- To *add* behavior but still use tox's implementation to install dependencies, implement this hook but do not return a value (or explicitly return `None`). One use-case may be to install (or ensure) non-python dependencies such as debian packages.
- To *override* tox's installation of dependencies, implement this hook and return a non-`None` value. One use-case may be to install via a different installation tool such as pip-accel or pip-faster.

Note: This api is experimental due to the unstable api of *`tox.venv.VirtualEnv`*.

Note: This hook uses `firstresult=True` (see pluggy first result only) – hooks implementing this will be run until one returns non-`None`.

class `tox.config.`**`Parser`**
Command line and ini-parser control object.

`add_argument`(**args*, ***kwargs*)
add argument to command line parser. This takes the same arguments that `argparse.ArgumentParser.add_argument`.

`add_testenv_attribute`(*name*, *type*, *help*, *default=None*, *postprocess=None*)
add an ini-file variable for "testenv" section.

Types are specified as strings like "bool", "line-list", "string", "argv", "path", "argvlist".

The `postprocess` function will be called for each testenv like `postprocess(testenv_config=testenv_config, value=value)` where `value` is the value as read from the ini (or the default value) and `testenv_config` is a *`tox.config.TestenvConfig`* instance which will receive all ini-variables as object attributes.

Any postprocess function must return a value which will then be set as the final value in the testenv section.

`add_testenv_attribute_obj`(*obj*)
add an ini-file variable as an object.

This works as the `add_testenv_attribute` function but expects "name", "type", "help", and "postprocess" attributes on the object.

class `tox.config.`**`Config`**
Global Tox config object.

`envconfigs = None`
Mapping envname -> envconfig

class `tox.config.`**`TestenvConfig`**
Testenv Configuration object.

In addition to some core attributes/properties this config object holds all per-testenv ini attributes as attributes, see "tox –help-ini" for an overview.

`config = None`
global tox config object

`envname = None`
test environment name

`envpython`
Path to python executable.

factors = None
: set of factors

get_envbindir()
: Path to directory where scripts/binaries reside.

get_envpython()
: path to python/jython executable.

get_envsitepackagesdir()
: Return sitepackagesdir of the virtualenv environment.

  NOTE: Only available during execution, not during parsing.

missing_subs = None
: Holds substitutions that could not be resolved.

  Pre 2.8.1 missing substitutions crashed with a ConfigError although this would not be a problem if the env is not part of the current testrun. So we need to remember this and check later when the testenv is actually run and crash only then.

python_info
: Return sitepackagesdir of the virtualenv environment.

class `tox.venv.`**VirtualEnv**

getcommandpath(*name*, *venv=True*, *cwd=None*)
: Return absolute path (str or localpath) for specified command name.

  - If it's a local path we will rewrite it as as a relative path.
  - If venv is True we will check if the command is coming from the venv or is whitelisted to come from external.

name
: test environment name.

path
: Path to environment base dir.

update(*action*)
: return status string for updating actual venv to match configuration. if status string is empty, all is ok.

class `tox.session.`**Session**
: The session object that ties together configuration, reporting, venv creation, testing.

getvenv(*name*)
: return a VirtualEnv controler object for the 'name' env.

installpkg(*venv*, *path*)
: Install package in the specified virtual environment.

  Parameters

  - **venv** (*VenvConfig*) – Destination environment
  - **path** (*str*) – Path to the distribution package.

  Returns True if package installed otherwise False.

  Return type bool

runenvreport(*venv*)
: Run an environment report to show which package versions are installed in the venv

5.9 Developers FAQ

This section contains information for users who want to extend the tox source code.

5.9.1 PyCharm

1. To generate the **project interpreter** you can use `tox -rvvve dev`.
2. For tests we use **pytest**, therefore change the Default test runner to `pytest`.
3. In order to be able to **debug** tests which create a virtual environment (the ones in `test_z_cmdline.py`) one needs to disable the PyCharm feature Attach to subprocess automatically while debugging (because virtualenv creation calls via subprocess to the `pip` executable, and PyCharm rewrites all calls to Python interpreters to attach to its debugger - however, this rewrite for pip makes it to have bad arguments: `no such option --port`).

5.9.2 Multiple Python versions on Windows

In order to run the unit tests locally all Python versions enlisted in `tox.ini` need to be installed.

Note: For a nice Windows terminal take a look at cmder.

One solution for this is to install the latest conda, and then install all Python versions via conda envs. This will create separate folders for each Python version.

```
conda create -n python2.7 python=2.7 anaconda
```

For tox to find them you'll need to:

- add the main installation version to the systems `PATH` variable (e.g. `D:\Anaconda` - you can use patheditor2)
- for other versions create a BAT scripts into the main installation folder to delegate the call to the correct Python interpreter:

  ```
  @echo off
  REM python2.7.bat
  @D:\Anaconda\pkgs\python-2.7.13-1\python.exe %*
  ```

This way you can also directly call from cli the matching Python version if you need to(similarly to UNIX systems), for example:

```
python2.7 main.py
python3.6 main.py
```

5.10 Writing a json result file

You can instruct tox to write a json-report file via:

```
tox --result-json=PATH
```

This will create a json-formatted result file using this schema:

```
{
  "testenvs": {
    "py27": {
      "python": {
        "executable": "/home/hpk/p/tox/.tox/py27/bin/python",
        "version": "2.7.3 (default, Aug  1 2012, 05:14:39) \n[GCC 4.6.3]",
        "version_info": [ 2, 7, 3, "final", 0 ]
      },
      "test": [
        {
          "output": "...",
          "command": [
            "/home/hpk/p/tox/.tox/py27/bin/pytest",
            "--instafail",
            "--junitxml=/home/hpk/p/tox/.tox/py27/log/junit-py27.xml",
            "tests/test_config.py"
          ],
          "retcode": "0"
        }
      ],
      "setup": []
    }
  },
  "platform": "linux2",
  "installpkg": {
    "basename": "tox-1.6.0.dev1.zip",
    "sha256": "b6982dde5789a167c4c35af0d34ef72176d0575955f5331ad04aee9f23af4326",
    "md5": "27ead99fd7fa39ee7614cede6bf175a6"
  },
  "toxversion": "1.6.0.dev1",
  "reportversion": "1"
}
```

5.11 Less announcing, more change-logging

With version 2.5.0 we dropped creating special announcement documents and rely on communicating all relevant changes through the CHANGELOG. See at pypi for a rendered version of the last changes containing links to the important issues and pull requests that were integrated into the release.

The historic release announcements are still online here for various versions:

- 0.5,
- 1.0,
- 1.1,
- 1.2,

- 1.3,
- 1.4,
- 1.4.3,
- 1.8,
- 1.9,
- 2.0,
- 2.4.0.

Happy testing, The tox maintainers

Python Module Index

Index