tox Documentation

Release 2.3.1

holger krekel

October 13, 2016

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` or `easy_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 = py26,py27
[testenv]
deps=pytest       # install pytest in the venvs
commands=py.test  # or 'nosetests' or ...
```

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 Python2.6 and Python2.7, just type:

```
tox
```

and watch things happening (you must have python2.6 and python2.7 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

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 py.test, nose or unittests in a uniform manner
- (new in 2.0) 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-2.6, 2.7, 3.2 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*

4.1 tox installation

4.1.1 Install info in a nutshell

Pythons: CPython 2.6-3.3, Jython-2.5.1, pypy-1.9ff

Operating systems: Linux, Windows, OSX, Unix

Installer Requirements: setuptools

License: MIT license

hg repository: https://bitbucket.org/hpk42/tox

4.1.2 Installation with pip/easy_install

Use one of the following commands:

```
pip install tox
easy_install tox
```

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

4.1.3 Install from Checkout

Consult the Bitbucket page to get a checkout of the mercurial repository:

https://bitbucket.org/hpk42/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.

4.2 tox configuration and usage examples

4.2.1 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 = py26,py27
[testenv]
deps=pytest # or 'nose' or ...
commands=py.test  # 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 py26
```

you can run restrict the test run to the python2.6 environment.

Available "default" test environments names are:

```
py
py24
py25
py26
py27
py30
py31
py32
py33
py34
jython
pypy
pypy3
```

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.

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:

```
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

New in version 1.6.1.

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

```
deps = -rrequirements.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` to exist at `{toxinidir}/requirements.txt`.

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 http://pypi.testrun.org
```

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

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

```
[tox]
indexserver =
    default = http://pypi.testrun.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 = http://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 `http://mypypiserver.org` url. You can override config file settings from the command line like this:

```
tox -i DEV=http://pypi.python.org/simple  # changes :DEV: package URLs
tox -i http://pypi.python.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 `easy_install` instead of pip:

```
[testenv]
install_command = easy_install {opts} {packages}
```

Or 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 http://packages.example.com --no-index {opts} {packa
```

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:

```
[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:

```
[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:

```
[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 setuptools/distribute test commands

Distribute/Setuptools support test requirements and you can extend its test command to trigger a test run when `python setup.py test` is issued:

```
from setuptools.command.test import test as TestCommand
import sys

class Tox(TestCommand):
    user_options = [('tox-args=', 'a', "Arguments to pass to tox")]
    def initialize_options(self):
        TestCommand.initialize_options(self)
        self.tox_args = None
    def finalize_options(self):
        TestCommand.finalize_options(self)
        self.test_args = []
        self.test_suite = True
    def run_tests(self):
        #import here, cause outside the eggs aren't loaded
        import tox
        import shlex
        args = self.tox_args
        if args:
            args = shlex.split(self.tox_args)
        errno = tox.cmdline(args=args)
        sys.exit(errno)

setup(
    #...,
    tests_require=['tox'],
    cmdclass = {'test': Tox},
    )
```

Now if you run:

```
python setup.py test
```

this will install tox and then run tox. You can pass arguments to `tox` using the `--tox-args` or `-a` command-line options. For example:

```
python setup.py test -a "-epy27"
```

is equivalent to running `tox -epy27`.

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{26,27,33}-django{15,16}-{sqlite,mysql}

[testenv]
deps =
    django15: Django>=1.5,<1.6
    django16: Django>=1.6,<1.7
    py33-mysql: PyMySQL     ; use if both py33 and mysql are in an env name
    py26,py27: urllib3      ; use if any of py26 or py27 are in an env name
    py{26,27}-sqlite: mock  ; mocking sqlite in python 2.x
```

4.2.2 py.test and tox

It is easy to integrate py.test 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 = py26,py31

[testenv]
deps=pytest       # PYPI package providing py.test
commands=
  py.test \
        {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 `python2.6` and `python3.1` 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 = py26,py31
[testenv]
changedir=tests
deps=pytest
commands=
  py.test \
    --basetemp={envtmpdir}  \ # py.test 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 `py.test` 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

`py.test` 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=
    py.test \
        --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 py.test run. Try to not use the "–basetemp" parameter.

installed-versus-checkout version. `py.test` 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 `py.test` invocation may end up importing the package from the checkout directory rather than the installed package.

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.

4.2.3 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 = py25,py26,py27

[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:

hg clone https://code.google.com/p/mock/

the checkout has a tox.ini that looks like this:

```
[tox]
envlist = py24,py25,py26,py27

[testenv]
deps=unittest2
commands=unit2 discover []

[testenv:py26]
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 2.4, 2.5, 2.6 and 2.7 respectively. Against Python2.6 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 *substitutions for positional arguments in commands* 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.

4.2.4 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 `python2.6` and `python2.5` 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.

4.2.5 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 `py.test`:

```
# in the testenv or testenv:NAME section of your tox.ini
commands =
    py.test {posargs}
```

or using `nosetests`:

```
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:

```
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]
basepython=python
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[,ENV2,...]` option you explicitely 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 py25,py26
```

which would run the commands of the `py25` and `py26` testenvironments respectively. The special value `ALL` selects all environments.

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

```
[tox]
envlist = py25,py26
```

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

```
export TOXENV=py25,py26 # 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

By default, for any `pyXY` test environment name the underlying "pythonX.Y" executable will be searched in your system `PATH`. It 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.

For `jython` and `pypy` the respective `jython` and `pypy-c` names will be looked for.

You can override any of the default settings by defining the `basepython` variable in a specific test environment section, for example:

```
[testenv:py27]
basepython=/my/path/to/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
    py.test
```

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 perfoms 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`.

4.2.6 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
    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 `py.test` it is done like this:

```
commands = py.test --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 neccessary 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 accomodate this issue one solution is to have `py.test` 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 whereever you keep your sphinx-docs
deps=sphinx
    py
commands=
    py.test --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 `py.test` 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.

4.2.7 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:

```
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
```

4.3 tox configuration specification

`tox.ini` files uses the standard ConfigParser "ini-style" format. Below you find the specification, but you might want to skim some tox configuration and usage examples first and use this page as a reference.

4.3.1 Tox global settings

List of optional global options:

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

skip_missing_interpreters=BOOL
New in version 1.7.2.

Setting this to `True` is equivalent of passing the `--skip-missing-interpreters` command line option, and 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 invokation. **Default:** `False`

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`

4.3.2 Virtualenv test environment settings

Test environments are defined by a:

```
[testenv:NAME]
...
```

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

```
[testenv]
...
```

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. **default**: interpreter used for tox invocation.

commands=ARGVLIST
the commands to be called for testing. Each command is defined by one or more lines; a command can have multiple lines if a line ends with the `\` character in which case the subsequent line will be appended (and may contain another `\` character ...). For eventually performing a call to `subprocess.Popen(args, ...)` `args` are determined by splitting the whole command by whitespace. Similar to `make` recipe lines, any command with a leading `-` will ignore the exit code.

install_command=ARGV
New in version 1.6.

the `install_command` setting is used for installing packages into the virtual environment; both the package under test and any defined dependencies. Must contain the substitution key `{packages}` which will be replaced by the packages to install. You should also accept `{opts}` if you are using pip or easy_install – it will contain index server options if you have configured them via *`indexserver`* and the deprecated `downloadcache` option if you have configured it.

default:

```
pip install {opts} {packages}
```

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. 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:

```
deps = :myindexserver:pkg
```

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

platform=REGEX
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.

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.

Note that the PATH, LANG and PIP_INDEX_URL variables are unconditionally passed down and on Windows SYSTEMROOT, PATHEXT, TEMP and TMP will be passed down as well whereas on unix TMPDIR will be passed down. 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.

recreate=True|False(default)
Always recreate virtual environment if this option is True.

downloadcache=path
DEPRECATED – as of August 2013 this option is not very useful because of pypi's CDN and because of caching pypi server solutions like devpi.

use this directory for caching downloads. This value is overriden by the environment variable PIP_DOWNLOAD_CACHE if it exists. If you specify a custom install_command that uses an installer other than pip, your installer must support the *–download-cache* command-line option. **default**: no download cache will be used.

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.

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 = http://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`

4.3.3 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.

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 and empty string if the environment variable does not exist.

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 refered 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}
```

4.3.4 Generating environments, conditional settings

New in version 1.8.

Suppose you want to test your package against python2.6, python2.7 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 = {py26,py27}-django{15,16}

[testenv]
basepython =
    py26: python2.6
    py27: python2.7
deps =
    pytest
    django15: Django>=1.5,<1.6
    django16: Django>=1.6,<1.7
    py26: unittest2
commands = py.test
```

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 = {py26,py27}-django{15,16}
```

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

```
py26-django15
py26-django16
py27-django15
py27-django16
```

You can still list environments explicitly along with generated ones:

```
envlist = {py26,py27}-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
py26-django15
py26-django16
py27-django15
py27-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:

```
basepython =
    py26: python2.6
    py27: python2.7
```

This conditional setting will lead to either `python2.6` or `python2.7` used as base python, e.g. `python2.6` is selected if current environment contains `py26` factor.

In list settings such as `deps` or `commands` you can freely intermix optional lines with unconditional ones:

```
deps =
    pytest
    django15: Django>=1.5,<1.6
    django16: Django>=1.6,<1.7
    py26: 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 2.6 environments.

Note: Tox provides good defaults for basepython setting, so the above ini-file can be further reduced by omitting the `basepython` setting.

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{26,27,33}-django{15,16}-{sqlite,mysql}

[testenv]
deps =
    py33-mysql: PyMySQL     ; use if both py33 and mysql are in an env name
    py26,py27: urllib3      ; use if any of py26 or py27 are in an env name
    py{26,27}-sqlite: mock  ; mocking sqlite in python 2.x
```

Take a look at first `deps` line. It shows how you can special case something for a combination of factors, you just join combining factors with a hyphen. This particular line states that `PyMySQL` will be loaded for python 3.3, mysql environments, e.g. `py33-django15-mysql` and `py33-django16-mysql`.

The second line shows how you use same line for several factors - by listing them delimited by commas. It's possible to list not only simple factors, but also their combinations like `py26-sqlite,py27-sqlite`.

Finally, factor expressions are expanded the same way as envlist, so last example could be rewritten as `py{26,27}-sqlite`.

Note: Factors don't do substring matching against env name, instead every hyphenated expression is split by - and if ALL the factors in an expression are also factors of an env then that condition is considered hold.

For example, environment `py26-mysql`:

- could be matched with expressions `py26`, `py26-mysql`, `mysql-py26`,
- but not with `py2` or `py26-sql`.

4.4 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.

4.5 V2: new tox multi-dimensional, platform-specific configuration

---

Note: This is a draft document sketching a to-be-done implementation. It does not fully specify each change yet but should give a good idea of where things are heading. For feedback, mail the testing-in-python mailing list or open a pull request on https://bitbucket.org/hpk42/tox/src/84d8cf3c2a95fefd874f22c8b2d257e94365472f/doc/config-v2.txt?at=default

---

Abstract: Adding multi-dimensional configuration, platform-specification and multiple installers to tox.ini.

Target audience: Developers using or wanting to use tox for testing their python projects.

4.6 Issues with current tox (1.4) configuration

Tox is used as a tool for creating and managing virtualenv environments and running tests in them. As of tox-1.4 there are some issues frequently coming up with its configuration language:

- there is no way to instruct tox to parametrize testenv specifications other than to list all combinations by specifying a `[testenv:...]` section for each combination. Examples of real life situations arising from this:
  - https://github.com/tomchristie/django-rest-framework/blob/b001a146d73348af18cfc4c943d87f2f389349c9/tox.ini
  - https://bitbucket.org/tabo/django-treebeard/src/93b579395a9c/tox.ini
- there is no way to have platform specific settings other than to define specific testenvs and invoke tox with a platform-specific testenv list.
- there is no way to specify the platforms against which a project shall successfully run.
- tox always uses pip for installing packages currently. This has several issues:
  - no way to check if installing via easy_install works
  - no installs of packages with compiled c-extensions (win32 standard)

4.7 Goals, resolving those issues

This document discusses a possible solution for each of these issues, namely these goals:

- allow to more easily define and run dependency/interpreter variants with testenvs
- allow platform-specific settings
- allow to specify platforms against which tests should run
- allow to run installer-variants (easy_install or pip, xxx)
- try to mimick/re-use bash-style syntax to ease learning curve.

4.8 Example: Generating and selecting variants

Suppose you want to test your package against python2.6, python2.7 and on the windows and linux platforms. Today you would have to write down 2*2 = 4 `[testenv:*]` sections and then instruct tox to run a specific list of environments on each platform.

With tox-1.X you can directlys specify combinations:

```
# combination syntax gives 2 * 2 = 4 testenv names
#
envlist = {py26,py27}-{win,linux}

[testenv]
deps = pytest
platform =
       win: windows
       linux: linux
basepython =
       py26: python2.6
       py27: python2.7
commands = py.test
```

Let's go through this step by step:

```
envlist = {py26,py27}-{windows,linux}
```

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

```
py26-windows
py26-linux
py27-windows
py27-linux
```

Our `[testenv]` uses a new templating style for the `platform` definition:

```
platform=
       windows: windows
       linux: linux
```

These two conditional settings will lead to either `windows` or `linux` as the platform string. When the test environment is run, its platform string needs to be contained in the string returned from `platform.platform()`. Otherwise the environment will be skipped.

The next configuration item in the `testenv` section deals with the python interpreter:

```
basepython =
       py26: python2.6
       py27: python2.7
```

This defines a python executable, depending on if `py26` or `py27` appears in the environment name.

The last config item is simply the invocation of the test runner:

```
commands = py.test
```

Nothing special here :)

Note: Tox provides good defaults for platform and basepython settings, so the above ini-file can be further reduced:

```
[tox]
envlist = {py26,py27}-{win,linux}

[testenv]
deps = pytest
commands = py.test
```

Voila, this multi-dimensional `tox.ini` configuration defines 2*2=4 environments.

4.9 The new "platform" setting

A testenv can define a new `platform` setting. If its value is not contained in the string obtained from calling `sys.platform` the environment will be skipped.

4.10 Expanding the `envlist` setting

The new `envlist` setting allows to use `{}` bash-style expressions. XXX explanation or pointer to bash-docs

4.11 Templating based on environments names

For a given environment name, all lines in a testenv section which start with "NAME: ..." will be checked for being part in the environment name. If they are part of it, the remainder will be the new line. If they are not part of it, the whole line will be left out. Parts of an environment name are obtained by −-splitting it.

Variant specification with [variant:VARNAME]

4.12 Showing all expanded sections

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 [XXX output ommitted for now]
```

4.13 Making sure your packages installs with easy_install

The new "installer" testenv setting allows to specify the tool for installation in a given test environment:

```
[testenv]
installer =
    easy: easy_install
    pip: pip
```

If you want to have your package installed with both easy_install and pip, you can list them in your envlist likes this:

```
[tox]
envlist = py[26,27,32]-django[13,14]-[easy,pip]
```

If no installer is specified, `pip` will be used.

4.14 Default settings related to environments names/variants

tox comes with predefined settings for certain variants, namely:

- `{easy,pip}` use easy_install or pip respectively
- `{py24,py25,py26,py27,py31,py32,py33,py34,pypy19]` use the respective pythonNN or PyPy interpreter
- `{win32,linux,darwin}` defines the according `platform`.

You can use those in your "envlist" specification without the need to define them yourself.

4.15 Use more bash-style syntax

tox leverages bash-style syntax if you specify mintoxversion = 1.4:

- $VARNAME or ${...} syntax instead of the older {} substitution.
- XXX go through config.txt and see how it would need to be changed

4.16 Transforming the examples: django-rest

The original django-rest-framework tox.ini file has 159 lines and a lot of repetition, the new one would +have 20 lines and almost no repetition:

```
[tox]
envlist = {py25,py26,py27}-{django12,django13}{,-example}

[testenv]
deps=
    coverage==3.4
    unittest-xml-reporting==1.2
    Pyyaml==3.10
    django12: django==1.2.4
    django13: django==1.3.1
    # some more deps for running examples
    example: wsgiref==0.1.2
    example: Pygments==1.4
    example: httplib2==0.6.0
    example: Markdown==2.0.3

commands =
   !example: python setup.py test
   example: python examples/runtests.py
```

Note that `{,-example}` in the envlist denotes two values, an empty one and a `example` one. The empty value means that there are no specific settings and thus no need to define a variant name.

4.17 Transforming the examples: django-treebeard

Another tox.ini has 233 lines and runs tests against multiple Postgres and Mysql engines. It also performs backend-specific test commands, passing different command line options to the test script. With the new tox-1.X we not only can do the same with 32 non-repetive configuration lines but we also produce 36 specific testenvs with specific dependencies and test commands:

```
[tox]
envlist =
 {py24,py25,py26,py27}-{django11,django12,django13}-{nodb,pg,mysql}, docs

[testenv:docs]
changedir = docs
deps =
    Sphinx
    Django
commands =
    make clean
    make html

 [testenv]
 deps=
       coverage
       pysqlite
       django11: django==1.1.4
       django12: django==1.2.7
       django13: django==1.3.1
       django14: django==1.4
```

```
      nodb: pysqlite
      pg: psycopg2
      mysql: MySQL-python

commands =
    nodb: {envpython} runtests.py {posargs}
    pg: {envpython} runtests.py {posargs} \
                    --DATABASE_ENGINE=postgresql_psycopg2 \
                    --DATABASE_USER=postgres {posargs}
    mysql: {envpython} runtests.py --DATABASE_ENGINE=mysql \
                                   --DATABASE_USER=root {posargs}
```

4.18 support and contact channels

Getting in contact:

- join the Testing In Python (TIP) mailing list for general and tox/test-tool interaction questions.
- file a report on the issue tracker
- hang out on the irc.freenode.net #pylib channel
- clone the mercurial repository and submit patches
- the tetamap blog, holger's twitter presence or for private inquiries holger krekel at gmail.

4.18.1 professional support

Note: Upcoming: professional testing with pytest and tox , 24th-26th June 2013, Leipzig.

If you are looking for on-site teaching or consulting support, contact holger at merlinux.eu, an association of experienced well-known Python developers.

4.19 Changelog history

4.19.1 2.3.1

- fix issue294: re-allow cross-section substitution for setenv.

4.19.2 2.3.0

- 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 issue285: 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 issue289: fix build_sphinx target, thanks Barry Warsaw.
- fix issue252: 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 guranteed.
- internal: push some optional object creation into tests because tox core doesn't need it.

4.19.3 2.2.1

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

4.19.4 2.2.0

- fix issue265 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 issue246: fix regression in config parsing by reordering such that {envbindir} can be used again in tox.ini. Thanks Olli Walsh.
- fix issue99: the {env:...} substitution now properly uses environment settings from the `setenv` section. Thanks Itxaka Serrano.
- fix issue281: make –force-deps work when urls are present in dependency configs. Thanks Glyph Lefkowitz for reporting.
- fix issue174: 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 issue280: properly skip missing interpreter if {envsitepackagesdir} is present in commands. Thanks BB:ceridwenv

4.19.5 2.1.1

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

4.19.6 2.1.0

- fix issue258, fix issue248, fix issue253: 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 issue259: 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.

4.19.7 2.0.2

- fix issue247: 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.

4.19.8 2.0.1

- fix wheel packaging to properly require argparse on py26.

4.19.9 2.0.0

- (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 issue233: avoid hanging with tox-setuptools integration example. Thanks simonb.
- fix issue120: allow substitution for the commands section. Thanks Volodymyr Vitvitski.
- fix issue235: fix AttributeError with –installpkg. Thanks Volodymyr Vitvitski.
- tox has now somewhat pep8 clean code, thanks to Volodymyr Vitvitski.
- fix issue240: 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

4.19.10 1.9.2

- backout ability that –force-deps substitutes name/versions in requirement files due to various issues. This fixes issue228, fixes issue230, fixes issue231 which popped up with 1.9.1.

4.19.11 1.9.1

- use a file instead of a pipe for command output in "–result-json". Fixes some termination issues with python2.6.
- allow –force-deps to override dependencies in "-r" requirements files. Thanks Sontek for the PR.
- fix issue227: use "-m virtualenv" instead of "-mvirtualenv" to make it work with pyrun. Thanks Marc-Andre Lemburg.

4.19.12 1.9.0

- fix issue193: 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 issue199: 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 issue11: add a `skip_install` per-testenv setting which prevents the installation of a package. Thanks Julian Krause.
- fix issue124: ignore command exit codes; when a command has a "-" prefix, tox will ignore the exit code of that command
- fix issue198: fix broken envlist settings, e.g. {py26,py27}{-lint,}
- fix issue191: lessen factor-use checks

4.19.13 1.8.1

- fix issue190: 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.

4.19.14 1.8.0

- 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 issue148: remove "__PYVENV_LAUNCHER__" from os.environ when starting subprocesses. Thanks Steven Myint.
- fix issue152: 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.

4.19.15 1.7.2

- fix issue150: 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 issue59: 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 issue164: 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.

4.19.16 1.7.1

- fix issue162: don't list python 2.5 as compatibiliy/supported
- fix issue158 and fix issue155: windows/virtualenv properly works now: call virtualenv through "python -m virtualenv" with the same interpreter which invoked tox. Thanks Chris Withers, Ionel Maries Cristian.

4.19.17 1.7.0

- 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 issue132: removing zip_safe setting (so it defaults to false) to allow installation of tox via easy_install/eggs. Thanks Jenisys.
- fix issue126: 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 issue140.
- fix issue130: 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 issue129: 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 issue128: 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 issue105 – don't depend on an existing HOME directory from tox tests.

4.19.18 1.6.1

- fix issue119: {envsitepackagesdir} is now correctly computed and has a better test to prevent regression.
- fix issue116: 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 issue118: 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 issue117: 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 issue102: change to {toxinidir} when installing dependencies. this allows to use relative path like in "-rrequirements.txt".

4.19.19 1.6.0

- fix issue35: 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 issue91: 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 https://pypi.python.org.
- fix issue1: empty setup files are properly detected, thanks Anthon van der Neuth
- remove toxbootstrap.py for now because it is broken.
- fix issue109 and fix issue111: 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.

4.19.20 1.5.0

- fix issue104: 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 issue2 - add whitelist_externals to be used in `[testenv*]` sections, allowing to avoid warnings for commands such as `make`, used from the commands value.
- fix issue97 - allow substitutions to reference from other sections (thanks Krisztian Fekete)
- fix issue92 - 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

4.19.21 1.4.3

- use pip-script.py instead of pip.exe on win32 to avoid the lock exe file on execution issue (thanks Philip Thiem)
- introduce -ll–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 issue84 - 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.

4.19.22 1.4.2

- 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.

4.19.23 1.4.1

- fix issue41 better quoting on windows - you can now use "<" and ">" in deps specifications, thanks Chris Withers for reporting

4.19.24 1.4

- fix issue26 - no warnings on absolute or relative specified paths for commands
- fix issue33 - 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 issue29 - correctly point to pytest explanation for importing modules fully qualified
- fix issue32 - 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.

4.19.25 1.3

- fix: allow to specify wildcard filesystem paths when specifying dependencies such that tox searches for the highest version
- fix issue issue21: 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)

4.19.26 1.2

- 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 issue10: 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)

4.19.27 1.1

- fix issue5 - don't require argparse for python versions that have it
- fix issue6 - recreate virtualenv if installing dependencies failed
- fix issue3 - fix example on frontpage
- fix issue2 - 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

4.19.28 1.0

- move repository and toxbootstrap links to http://bitbucket.org/hpk42/tox
- fix issue7: introduce a "minversion" directive such that tox bails out if it does not have the correct version.
- fix issue24: introduce a way to set environment variables for for test commands (thanks Chris Rose)
- fix issue22: 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 issue20: document format of URLs for specifying dependencies
- fix issue19: 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

4.19.29 0.9

- fix pip-installation mixups by always unsetting PIP_RESPECT_VIRTUALENV (thanks Armin Ronacher)
- issue1: 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)

4.19.30 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

4.19.31 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 issue4: always prepend venv-path to PATH variable when calling subprocesses
- fix issue2: 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

4.19.32 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

4.19.33 0.5

- initial release

4.20 tox plugins

New in version 2.0.

With tox-2.0 a few aspects of tox running can be experimentally modified by writing hook functions. The list of of available hook function is to grow over time on a per-need basis.

4.20.1 writing a setuptools entrypoints plugin

If you have a `tox_MYPLUGIN.py` module you could use the following rough `setup.py` to make it into a package which you can upload to the Python packaging index:

```
# content of setup.py
from setuptools import setup

if __name__ == "__main__":
    setup(
        name='tox-MYPLUGIN',
        description='tox plugin decsription',
```

```
        license="MIT license",
        version='0.1',
        py_modules=['tox_MYPLUGIN'],
        entry_points={'tox': ['MYPLUGIN = tox_MYPLUGIN']},
        install_requires=['tox>=2.0'],
    )
```

If installed, the `entry_points` part will make tox see and integrate your plugin during startup.

You can install the plugin for development ("in-place") via:

```
pip install -e .
```

and later publish it via something like:

```
python setup.py sdist register upload
```

4.20.2 Writing hook implementations

A plugin module defines one or more hook implementation functions by decorating them with tox's `hookimpl` marker:

```
from tox import hookimpl

@hookimpl
def tox_addoption(parser):
    # add your own command line options


@hookimpl
def tox_configure(config):
    # post process tox configuration after cmdline/ini file have
    # been parsed
```

If you put this into a module and make it pypi-installable with the `tox` entry point you'll get your code executed as part of a tox run.

4.20.3 tox hook specifications and related API

Hook specifications for tox.

`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 have been parsed and the ini-file has been read. Please be aware that the config object layout may change as its API was not designed yet wrt to providing stability (it was an internal thing purely before tox-2.0).

`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_testenv_create`**(*venv*, *action*)
: [experimental] perform creation action for this venv.

tox.hookspecs.**tox_testenv_install_deps**(*venv*, *action*)
[experimental] perform install dependencies action for this venv.

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
dictionary containing envname to envconfig mappings

option = None
option namespace containing all parsed command line options

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. (only available during execution, not parsing)

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 localpath 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**
(unstable API). the session object that ties together configuration, reporting, venv creation, testing.

get_installpkg_path()

Returns Path to the distribution

Return type py.path.local

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

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

:param tox.config.VenvConfig: Destination environment :param str path: Path to the distribution package. :return: True if package installed otherwise False. :rtype: bool

4.21 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/py.test",
            "--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"
}
```


4.22 tox 0.5: a generic virtualenv and test management tool for Python

I have been talking about with various people in the last year and am happy to now announce the first release of `tox`. It aims to automate tedious Python related test activities driven from a simple `tox.ini` file, including:

- creation and management of different virtualenv environments
- installing your package into each of them
- running your test tool of choice (including e.g. running sphinx checks)
- testing packages against each other without needing to upload to PyPI

`tox` runs well on Python2.4 up until Python3.1 and integrates well with Continuous Integration servers Jenkins. There are many real-life examples and a good chunk of docs. Read it up on

http://codespeak.net/tox

and please report any issues. This is a fresh project and i'd like to drive further improvements from real world needs.

best,

holger krekel

4.23 tox 1.0: the rapid multi-python test automatizer

I am happy to announce tox 1.0, mostly a stabilization and streamlined release. TOX automates tedious test activities driven from a simple `tox.ini` file, including:

- creation and management of different virtualenv environments with different Python interpreters
- packaging and installing your package into each of them
- running your test tool of choice, be it nose, py.test or unittest2 or other tools such as "sphinx" doc checks
- testing dev packages against each other without needing to upload to PyPI

Docs and examples are at:

http://tox.readthedocs.org

Installation:

pip install -U tox

Note that code hosting and issue tracking has moved from Google to Bitbucket:

https://bitbucket.org/hpk42/tox

The 1.0 release includes contributions and is based on feedback and work from Chris Rose, Ronny Pfannschmidt, Jannis Leidel, Jakob Kaplan-Moss, Sridhar Ratnakumar, Carl Meyer and others. Many thanks!

best, Holger Krekel

4.23.1 CHANGES

- fix issue24: introduce a way to set environment variables for for test commands (thanks Chris Rose)
- fix issue22: require virtualenv-1.6.1, obsoleting virtualenv5 (thanks Jannis Leidel) and making things work with pypy-1.5 and python3 more seemlessly
- toxbootstrap.py (used by jenkins build slaves) now follows the latest release of virtualenv
- fix issue20: document format of URLs for specifying dependencies
- fix issue19: 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

4.24 tox 1.1: the rapid multi-python test automatizer

I am happy to announce tox 1.1, a bug fix release easing some commong workflows. TOX automates tedious test activities driven from a simple `tox.ini` file, including:

- creation and management of different virtualenv environments with different Python interpreters
- packaging and installing your package into each of them
- running your test tool of choice, be it nose, py.test or unittest2 or other tools such as “sphinx” doc checks
- testing dev packages against each other without needing to upload to PyPI

It works well on virtually all Python interpreters that support virtualenv.

Docs and examples are at:

http://tox.readthedocs.org

Installation:

pip install -U tox

Note that code hosting and issue tracking has moved from Google to Bitbucket:

https://bitbucket.org/hpk42/tox

The 1.0 release includes contributions and is based on feedback and work from Chris Rose, Ronny Pfannschmidt, Jannis Leidel, Jakob Kaplan-Moss, Sridhar Ratnakumar, Carl Meyer and others. Many thanks!

best, Holger Krekel

4.24.1 CHANGES

- fix issue5 - don't require argparse for python versions that have it
- fix issue6 - recreate virtualenv if installing dependencies failed
- fix issue3 - fix example on frontpage
- fix issue2 - 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 environents from a python2 virtualenv installation.

4.25 tox 1.2: the virtualenv-based test run automatizer

I am happy to announce tox 1.2, now using and depending on the latest virtualenv code and containing some bug fixes. TOX automates tedious test activities driven from a simple `tox.ini` file, including:

- creation and management of different virtualenv environments with different Python interpreters
- packaging and installing your package into each of them
- running your test tool of choice, be it nose, py.test or unittest2 or other tools such as "sphinx" doc checks
- testing dev packages against each other without needing to upload to PyPI

It works well on virtually all Python interpreters that support virtualenv.

Docs and examples are at:

> http://tox.testrun.org/

Installation:

> pip install -U tox

code hosting and issue tracking on bitbucket:

> https://bitbucket.org/hpk42/tox

best, Holger Krekel

4.25.1 1.2 compared to 1.1

- 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 inling tried to work around)
- fix issue10: work around UnicodeDecodeError when inokving 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 testevironment (thanks Michael Foord for reporting)

4.26 tox 1.3: the virtualenv-based test run automatizer

I am happy to announce tox 1.3, containing a few improvements over 1.2. TOX automates tedious test activities driven from a simple `tox.ini` file, including:

- creation and management of different virtualenv environments with different Python interpreters
- packaging and installing your package into each of them
- running your test tool of choice, be it nose, py.test or unittest2 or other tools such as "sphinx" doc checks
- testing dev packages against each other without needing to upload to PyPI

Docs and examples are at:

http://tox.testrun.org/

Installation:

pip install -U tox

code hosting and issue tracking on bitbucket:

https://bitbucket.org/hpk42/tox

best, Holger Krekel

4.26.1 1.3

- fix: allow to specify wildcard filesystem paths when specifiying dependencies such that tox searches for the highest version
- fix issue issue21: 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)

4.27 tox 1.4: the virtualenv-based test run automatizer

I am happy to announce tox 1.4 which brings:

- improvements with configuration file syntax, now allowing re-using selected settings across config file sections. see http://testrun.org/tox/latest/config.html#substitution-for-values-from-other-sections
- terminal reporting was simplified and streamlined. Now with verbosity==0 (the default), less information will be shown and you can use one or multiple "-v" options to increase verbosity.
- internal re-organisation so that the separately released "detox" tool can reuse tox code to implement a fully distributed tox run.

More documentation:

http://tox.testrun.org/

Installation:

pip install -U tox

code hosting and issue tracking on bitbucket:

https://bitbucket.org/hpk42/tox

4.27.1 What is tox?

tox standardizes and automates tedious test activities driven from a simple `tox.ini` file, including:

- creation and management of different virtualenv environments with different Python interpreters
- packaging and installing your package into each of them
- running your test tool of choice, be it nose, py.test or unittest2 or other tools such as "sphinx" doc checks
- testing dev packages against each other without needing to upload to PyPI

best, Holger Krekel

4.27.2 1.4

- fix issue26 - no warnings on absolute or relative specified paths for commands
- fix issue33 - 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 issue29 - correctly point to pytest explanation for importing modules fully qualified
- fix issue32 - 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 python-2.4, just tox itself requires 2.5 or higher.

4.28 tox 1.4.3: the Python virtualenv-based testing automatizer

tox 1.4.3 fixes some bugs and introduces a new script and two new options:

- "tox-quickstart" - run this script, answer a few questions, and get a tox.ini created for you (thanks Marc Abramowitz)
- "tox -l" lists configured environment names (thanks Lukasz Balcerzak)
- (experimental) "–installpkg=localpath" option which will skip the sdist-creation of a package and instead install the given localpath package.
- use pip-script.py instead of pip.exe on win32 to avoid windows locking the .exe

Note that the sister project "detox" should continue to work - it's a separately released project which drives tox test runs on multiple CPUs in parallel.

More documentation:

http://tox.testrun.org/

Installation:

pip install -U tox

repository hosting and issue tracking on bitbucket:

https://bitbucket.org/hpk42/tox

4.28.1 What is tox?

tox standardizes and automates tedious python driven test activities driven from a simple `tox.ini` file, including:

- creation and management of different virtualenv environments with different Python interpreters
- packaging and installing your package into each of them
- running your test tool of choice, be it nose, py.test or unittest2 or other tools such as "sphinx" doc checks
- testing dev packages against each other without needing to upload to PyPI

best, Holger Krekel

4.29 CHANGELOG

4.29.1 1.4.3 (compared to 1.4.2)

- introduce -ll–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 issue84 - 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 Warszaw for both.

4.30 tox 1.8: Generative/combinatorial environments specs

I am happy to announce tox 1.8 which implements parametrized environments.

See https://tox.testrun.org/latest/config.html#generating-environments-conditional-settings for examples and the new backward compatible syntax in your tox.ini file.

Many thanks to Alexander Schepanovski for implementing and refining it based on the specifcation draft.

More documentation about tox in general:

http://tox.testrun.org/

Installation:

pip install -U tox

code hosting and issue tracking on bitbucket:

https://bitbucket.org/hpk42/tox

4.30.1 What is tox?

tox standardizes and automates tedious test activities driven from a simple `tox.ini` file, including:

- creation and management of different virtualenv environments with different Python interpreters
- packaging and installing your package into each of them
- running your test tool of choice, be it nose, py.test or unittest2 or other tools such as "sphinx" doc checks
- testing dev packages against each other without needing to upload to PyPI

best, Holger Krekel, merlinux GmbH

4.30.2 Changes 1.8 (compared to 1.7.2)

- 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 issue148: remove "__PYVENV_LAUNCHER__" from os.environ when starting subprocesses. Thanks Steven Myint.
- fix issue152: 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.

4.31 tox-1.9: refinements, fixes (+detox-0.9.4)

tox-1.9 was released to pypi, a maintenance release with mostly backward-compatible enhancements and fixes. However, tox now defaults to pip-installing only non-development releases and you have to set "pip_pre = True" in your testenv section to have it install development ("pre") releases.

In addition, there is a new detox-0.9.4 out which allow to run tox test environments in parallel and fixes a compat problem with eventlet.

Thanks to Alexander Schepanosvki, Florian Schulze and others for the contributed fixes and improvements.

More documentation about tox in general:

http://tox.testrun.org/

Installation:

pip install -U tox

code hosting and issue tracking on bitbucket:

https://bitbucket.org/hpk42/tox

4.31.1 What is tox?

tox standardizes and automates tedious test activities driven from a simple `tox.ini` file, including:

- creation and management of different virtualenv environments with different Python interpreters
- packaging and installing your package into each of them
- running your test tool of choice, be it nose, py.test or unittest2 or other tools such as "sphinx" doc checks
- testing dev packages against each other without needing to upload to PyPI

best, Holger Krekel, merlinux GmbH

4.31.2 1.9.0

- fix issue193: 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 issue199: 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 issue11: add a `skip_install` per-testenv setting which prevents the installation of a package. Thanks Julian Krause.
- fix issue124: ignore command exit codes; when a command has a "-" prefix, tox will ignore the exit code of that command
- fix issue198: fix broken envlist settings, e.g. {py26,py27}{-lint,}
- fix issue191: lessen factor-use checks

4.32 tox-2.0: plugins, platform, env isolation

tox-2.0 was released to pypi, a major new release with *mostly* backward-compatible enhancements and fixes:

- experimental support for plugins, see https://testrun.org/tox/latest/plugins.html which includes also a refined internal registration mechanism for new testenv ini options. You can now ask tox which testenv ini parameters exist with `tox --help-ini`.
- ENV isolation: only pass through very few environment variables from the tox invocation to the test environments. This may break test runs that previously worked with tox-1.9 – you need to either use the `setenv` or `passenv` ini variables to set appropriate environment variables.
- PLATFORM support: you can set `platform=REGEX` in your testenv sections which lets tox skip the environment if the REGEX does not match `sys.platform`.
- tox now stops execution of test commands if the first of them fails unless you set `ignore_errors=True`.

Thanks to Volodymyr Vitvitski, Daniel Hahler, Marc Abramowitz, Anthon van der Neuth and others for contributions.

More documentation about tox in general:

> http://tox.testrun.org/

Installation:

> pip install -U tox

code hosting and issue tracking on bitbucket:

https://bitbucket.org/hpk42/tox

4.32.1 What is tox?

tox standardizes and automates tedious test activities driven from a simple `tox.ini` file, including:

- creation and management of different virtualenv environments with different Python interpreters
- packaging and installing your package into each of them
- running your test tool of choice, be it nose, py.test or unittest2 or other tools such as "sphinx" doc checks
- testing dev packages against each other without needing to upload to PyPI

best, Holger Krekel, merlinux GmbH

4.32.2 2.0.0

- (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 issue233: avoid hanging with tox-setuptools integration example. Thanks simonb.
- fix issue120: allow substitution for the commands section. Thanks Volodymyr Vitvitski.
- fix issue235: fix AttributeError with –installpkg. Thanks Volodymyr Vitvitski.
- tox has now somewhat pep8 clean code, thanks to Volodymyr Vitvitski.
- fix issue240: 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

t

A

B

C

D

E

F

G

I

N

O

P

R

S

T

U

V

W