CNRS ANF PYTHON

Packaging & Life Cycle

Marc Poinot
Numerical Simulation Dept.
marc.poinot@onera.fr

ONERA
THE FRENCH AEROSPACE LAB

retour sur innovation

Outline

Package management with Python

- Concepts
  - Software life cycle
  - Package services
  - Pragmatic approach
- Practical works
  - Source control system
  - Production & installation
  - Documentation
  - Test
  - From tools to process

How about yourself?

- You are looking for a service
  - Use google, Pypi, collegues, articles, existing software
  - Code by yourself
    - Technical, strategical, personnal reasons
- Use an already packaged module
  - Sometimes automated process
  - RPM, easy_install, Anaconda...
  - Easier on windows
- Use the source
  - Download, produce, install
  - May require privileges

When did you give up?

- Cannot find the module
  - That fulfill my requirements
  - Don't want the same as my neighbour
- Last modified date is too old...
  - 1 week, 1 month, 1 years
  - Inactive forum, mailing list
- Have to rebuild the system
  - Too much dependancies
  - Incompatible version/ productions
  - Cannot change host platform
- Get bored
  - Fail to build in less than 10 minutes, 1 hour, 1 day
  - No way to understand how to use it
  - Existing features not usable or not implemented yet

When did you give up?

- Cannot find the module
  - That fulfill my requirements
  - Don't want the same as my neighbour
- Last modified date is too old...
  - 1 week, 1 month, 1 years
  - Inactive forum, mailing list
- Have to rebuild the system
  - Too much dependancies
  - Incompatible version/ productions
  - Cannot change host platform
- Get bored
  - Fail to build in less than 10 minutes, 1 hour, 1 day
  - No way to understand how to use it
  - Existing features not usable or not implemented yet

Packaging & Life Cycle

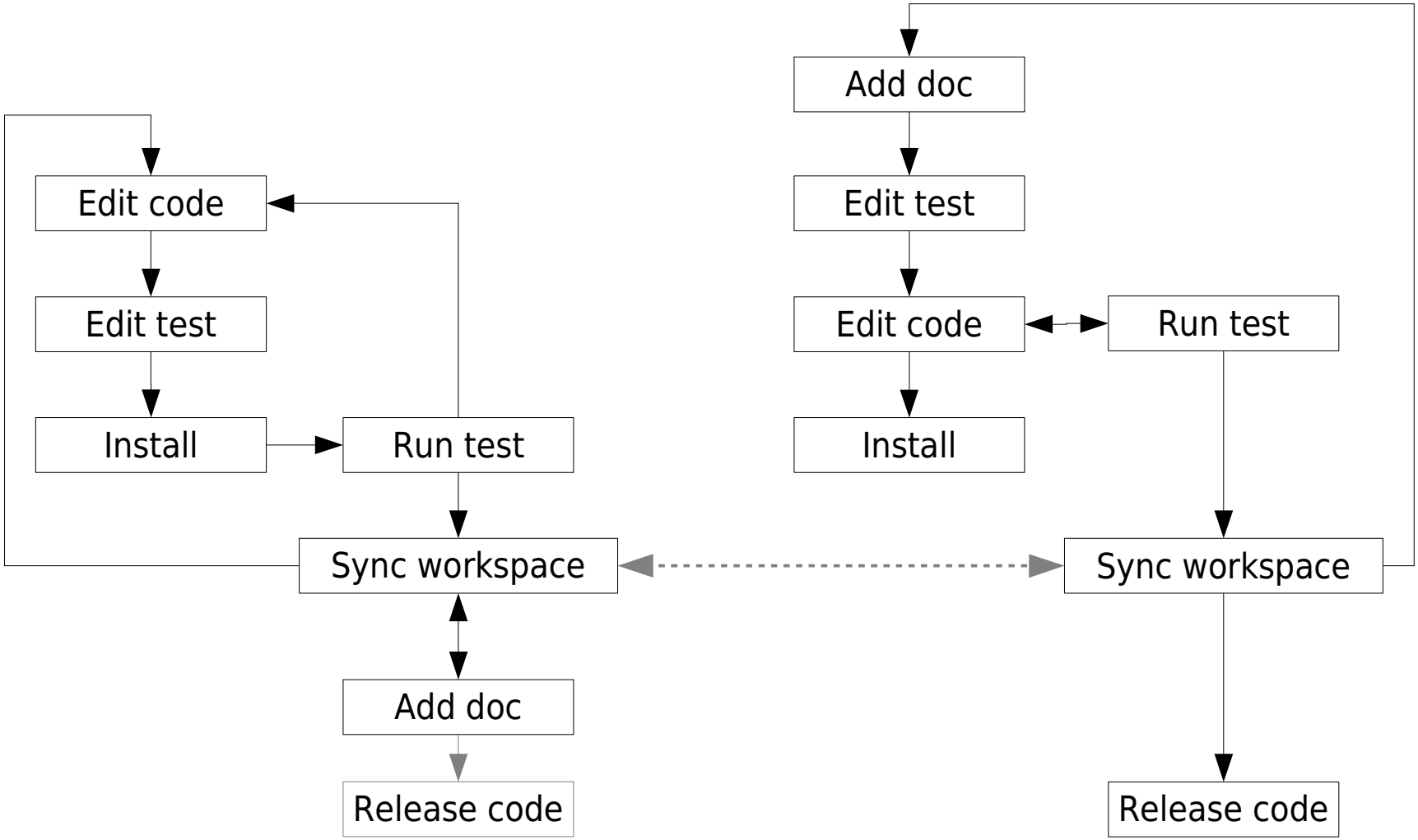
- Manage your software as you want others to do so
- Package
  - Easy to find, understand, install, use
- What are the key points?
  - Package life cycle
  - Source control system
  - Production & installation
  - Documentation
  - Test
  - Distribution

marc.poinot@onera.fr

Pros & Cons

- Pros
  - Better knowledge of your code
  - Separates know-how /pipes
  - Many python tools
  - Community as test team
  - Help back community
  - Increase productivity
  - Increase reusability
  - Fun
- Cons
  - No time to spend on that
  - Nobody else uses my code
  - Only a test
  - Clone of another code
  - Not fun

Life cycle example



marc.poinot@onera.fr

Production & Install - 1

- Concepts
  - Source/ Process/ Product
  - Store product and/ or process
  - Tools availability
  - Reproductibility/ Stability/ Dependancies
  - At least three targets: developpement, installation, test
- Processes
  - Generation SWIG, Cython, doc,
  - Compilation C, C++, Fortran
  - Quality checks, tests, perfs
  - Files selection copy/ install/ configuration
- Support the life cycle
  - Automated, Reproductible

Production & Install - 2

- Full python production
  - Use scripts
  - Provide configuration tips as Python
- Configuration as Python files
  - Platform identification & configuration
  - Dependancies description and/ or detection
- Distutils
  - Directory layout
  - setup.py file
  - __init__.py files

Production & Install - 3

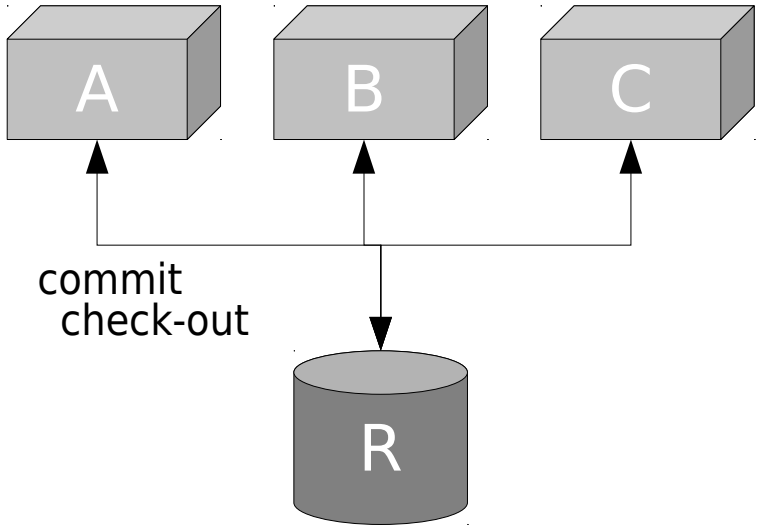
setup.py

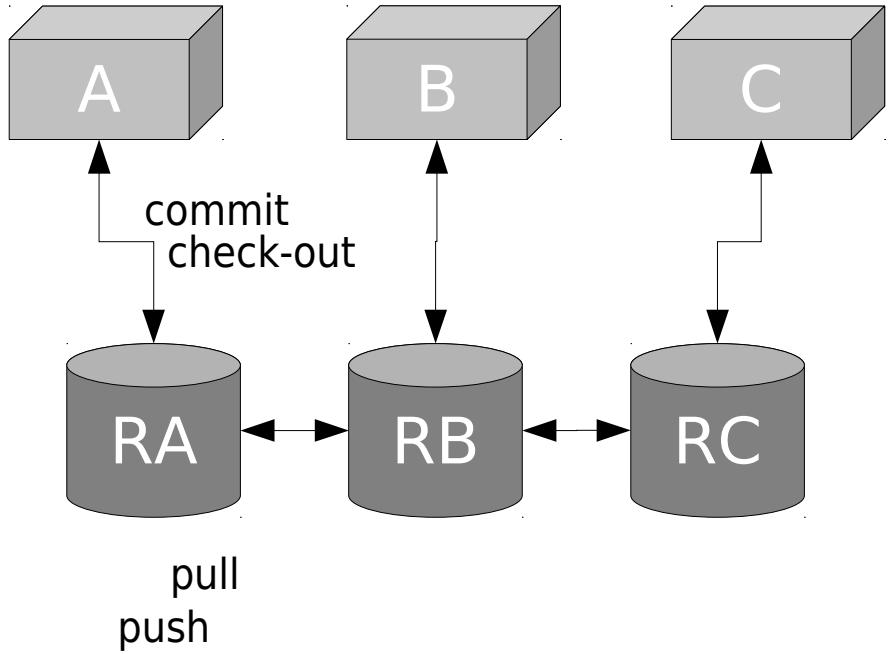
```
setup(
name          = "pyShift",
version       = "0.1",
description   = "Rigid Motion for 2D grids",
author        = "ONERA/DSNA/CS2A Marc Poinot",
author_email = "marc.poinot@onera.fr",
packages      = ['pyShift'],
ext_modules   = [Extension("pyShift.gengrid_stub",
                           ["pyShift/src/gengrid_stub.pyx",
                            "pyShift/src/gengrid.c"],
                           include_dirs = PATH_INCLUDES,
                           library_dirs = PATH_LIBRARIES,
                           libraries    = LINK_LIBRARIES,
                           )],
cmdclass      = {'build_ext':build_ext,'clean':clean},
)
```

Source control - 1

- Concepts
  - Repository
    - Central or local
    - Store source & modifications
    - Branches, change sets
    - Add stamps, tags, comments
    - Synchronize, merge, rollback
  - Workspace
    - Local copy, merge
    - Used as plain directory structure
    - Except add/ remove files/ dirs
  - Repository & Workspace synchronization

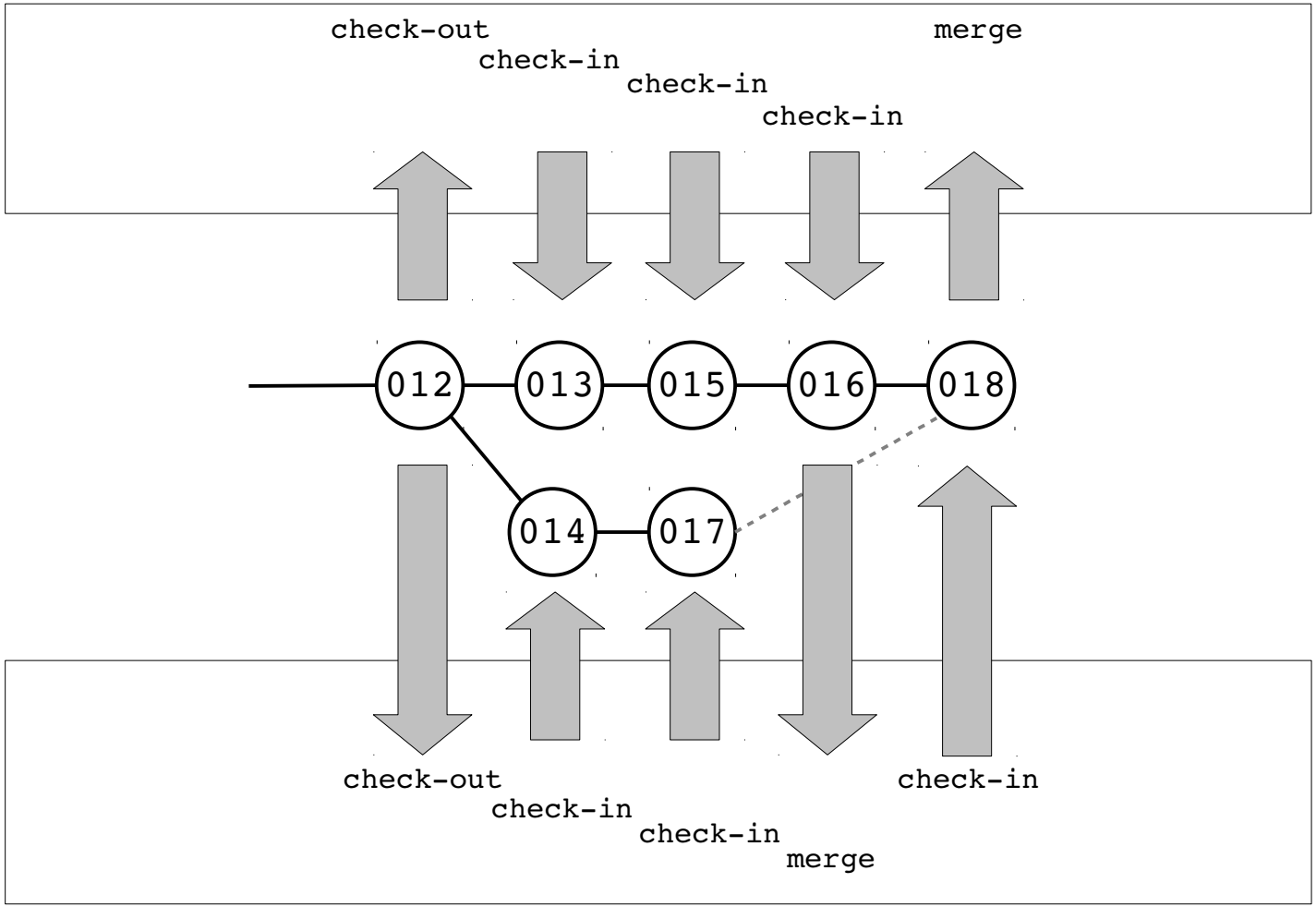
Source control - 2





marc.poinot@onera.fr

Source control - 3



marc.poinot@onera.fr

Documentation - 1

- End user documentation
  - Tutorial, examples
  - Reference guide
    - Should be generated from sources
  - Installation guide
    - Shortest is better
- Embedded into the package
  - Doc version with code version
    - Easy update
    - Local generation
- Device independant format
  - Targets HTML & LaTeX (PDF)

Documentation - 2

- Sphinx
  - Documentation generator from ReST
  - Python module doc auto generation
  - No generators for C/C++/Fortran
- Production
  - Re-run process at install time
    - Requires tools
  - Store .tar.html and pdf
    - Watch the source control

marc.poinot@onera.fr

Documentation - 3

Sphinx - conf.py

```
# should be run in root dir (i.e. setup.py as brother)
export RDIR=./.scons.linux2.tmp/build
mkdir -p $RDIR/doc/html
mkdir -p $RDIR/doc/html/images
cp $RDIR/src/include/CHLone/*.txt ./doc
sphinx-build -c doc -b html doc $RDIR/doc/html
sphinx-build -c doc -b latex doc $RDIR/doc/latex
cp doc/images/* $RDIR/doc/html/images
(cd $RDIR/doc/html; tar cvf ../../../CHLone-html.tar .)
(cd $RDIR/doc/latex; pdflatex CHLone.tex)
```

Documentation - 4

Sphinx - ReST

```
.. pyShift - copyright CNRS 2013

pyShift - Rigid Motion for 3D Grids
===================================

**pyShift** module features include a cartesian mesh generator and a rigid 3D
grid motion. Grids, or meshes, are 3D structured meshes with (``i``, ``j``, ``k``)
indexes. Available grid generators are:

 * square ([#N1]_)
 * rectangle
 * cube
 * parallepipède

The cartesian grid defines x,y,z points in a 3 dimensions (i,j,k) array.
Coordinates go from 1 to N for N points, N is the parameter for the grid
generation [#N2]_.

.. [#N1] topological dim is 2, physical dims is 3


Contents:

.. toctree::
   :maxdepth: 2

   Grid generation <gengrids>
   Rigid motion <motion>
   installation
   other

* :ref:`genindex`
* :ref:`search`
```

Test - 1

- Check actual features
  - End-user tests
    - services, no-regress
  - Internal tests
    - production, installation, prefs
- Black-box vs White box
  - Interface includes
    - Function services, args, return
    - Errors, exceptions, constants
    - Protocol
- Coverage
  - Test suites/ tests
  - Actual confidence in test suite

- End-user tightly related to documentation

Test - 2

- Nose
  - Layer on top of unittest
- Unittest
  - Default module
  - More complex to define/ use
  - White-box oriented
- Write tests first
  - Implementation is the mean to reach the test
- Write User manual first
  - Explain the interface and its use
  - Write examples to be run as tests

Test - 3

function

```
def test_gen():
  """Test mesh generation"""
  g1=GGN.parallelepiped(3,5,7)
  g2=GGN.cube(7)
  g3=GGN.rectangle(3,5)
  g4=GGN.square(5)
  return True
```


marc.poinot@onera.fr

Test - 4

doctest

```
def shift(g1,p0,p1,alpha,trans):
    """Mesh rotation on arbitrary axis.
    g1: the mesh (numpy array)
    p0,p1: two points for the rotation axis definition
    alpha: rotation angle (radian)
    trans: translation (x,y,z) tuple or list or ndarray
    returns g2 a new grid result of motion on g1 (g1 is unchanged)

    example:

    >>> import pyShift.gengrid_stub
    >>> import pyShift.motion
    >>> g0=pyShift.gengrid_stub.square(3)
    >>> g1=pyShift.motion.shift(g0,(0,0,0),(1,1,1),45.,(0,0,0))
    >>> g1.tolist()[0][0]
    [[0.0], [-0.3330433752167512], [-0.6660867504335024]]

    """
```


marc.poinot@onera.fr

Test - 5

unittest

```
class MotionTestCase(unittest.TestCase):
  def setUp(self):
    self.mesh=GGN.parallelepiped(3,5,7)
  def test_00Module(self):
    import pyShift.motion
  def test_01Rotate(self):
    alpha=45*(math.pi/180.)
    p0=(0.0,0.0,0.0)
    p1=(0.0,1.0,0.0)
    trans=(0.0,0.0,0.0)
    g1=MTN.shift(self.mesh,p0,p1,alpha,trans)
    self.assertFalse((g1[0][0]==self.mesh[0][0]).all())
    self.assertTrue((g1[1][0]==self.mesh[1][0]).all())
    self.assertFalse((g1[2][0]==self.mesh[2][0]).all())
```


marc.poinot@onera.fr

Package systems - 1

- Self-contained package
  - Description
  - Reference
    - Version, platform
  - Contents
    - Source, production process
    - Products (doc, default config files, ...)
  - Depends
    - Autodetection (*pip freeze*), autoconfiguration
    - Compatibility (require, provide, obsolete)
    - Uninstall
    - Shipped for a referenced framework (Anaconda 1.7)
    - Force local environment (virtualenv)

- Find a package
    - Repository
      - Pypi

Package systems - 2

- Create package for a package management systems
  - Linux based
    - RPM, Portage, YUM
  - Windows based
    - Inno, NSIS, windows install
  - Python based
    - tools: Distutils, setup-tools (easy_install), pip
    - format: egg, wheel

Practical works

- Start with pySHIFT
  - You have a set of files
  - You make to release it as a package
- Steps
  - 1 - Production & installation
  - 2 - Source control
  - 3 - Documentation
  - 4 - Test
  - 5 - Shipping

Step 1 - Production & Installation - 1

- Distutils tools
  - Write a setup.py

    ```
    python setup.py build
    python setup.py install --prefix=/tmp/install
    ```
  - Change files/ directories hierarchy
  - Add platform detection, fortran, cython production
  - Run display test

    ```
    python -c 'import pyShift.display;pyShift.display.test()'
    ```

Step 1 - Production & Installation - 2

```
from distutils.core import setup,Extension
from Cython.Distutils import build_ext
import numpy
PATH_INCLUDES=[numpy.get_include()]
PATH_LIBRARIES=['pyShift/lib']
LINK_LIBRARIES=['gen3d']
setup(
name          = "pyShift",
version       = "0.1",
packages      = ['pyShift'],
ext_modules   = [Extension("pyShift.gengrid_stub",
                           ["pyShift/src/gengrid_stub.pyx",
                            "pyShift/src/gengrid.c"],
                            include_dirs = PATH_INCLUDES,
                            library_dirs = PATH_LIBRARIES,
                            libraries    = LINK_LIBRARIES,
                           )],
cmdclass      = {'build_ext':build_ext},
```


marc.poinot@onera.fr

Step 2 - Source control - 1

- Mercurial
  - Add project
  - Change add/ remove files
  - Merge with student next to you
    - hg init
    - hg add
    - hg remove
    - hg commit
    - hg tag
    - hg merge
    - hg update
    - hg pull
    - hg push
    - hg diff
- Clear test project
  - Start a new one next step

Step 2 - Source Control - 2

```
mv mycode pyShift-v0.1
cd pyShift-v0.1
hg init
vi .hgignore
vi readme.txt
mkdir doc test
vi doc/index.txt
vi test/run.py
hg add doc test
hg commit -m 'First integration'
hg pull
hg update
hg commit -m 'Merge from student next to me'
hg tag v0.1
hg archive ../pyShift-v0.1.tar.gz
```

```
syntax: glob
*~
*.pyc
build/
```

```
pyShift - Licence LGPL v2
```

```
pyShift
-------

Grid rigid motion module
```

```
pass
```


marc.poinot@onera.fr

Step 3 - Documentation

- Sphinx
  - Add docs
    - User guide
    - Refererence guide
    - Installation
  - Multiple sources
    - python/ cython
    - fortran
    - plain text
- Produce doc
- Update source control

Step 4 - Test

- Test framework
  - Do not run test suite in the module directory

```
python setup.py install
--prefix=$INSTALL
--single-version-externally-managed --root=/
```

  - Unittest
  - Nose (with coverage module)

```
nose -w $SITEPACKAGE/pyShift
--with-coverage --cover-package=pyShift
--with-doctest
```

- Test process
  - Document/ Write test/ Code/ Install/ Test report
- Update source control

Step 5 - Shipping

- Package file
  - Use setuptools instead of setup
  - Add requirements

    ```
    python setup.py sdist
    ```
  - Anaconda
    - Use Conda tools

    ```
    meta.yaml build.sh
    conda build .
    ```
  - Mercurial

    ```
    hg archive
    ```
- Update source control

Conclusion

- Packaging Python modules
  - Lot of Python tools
  - Find your own way...
    - ...But find a way
  - Not that difficult
- Be in the community
  - You are using Open Source
  - Give help back