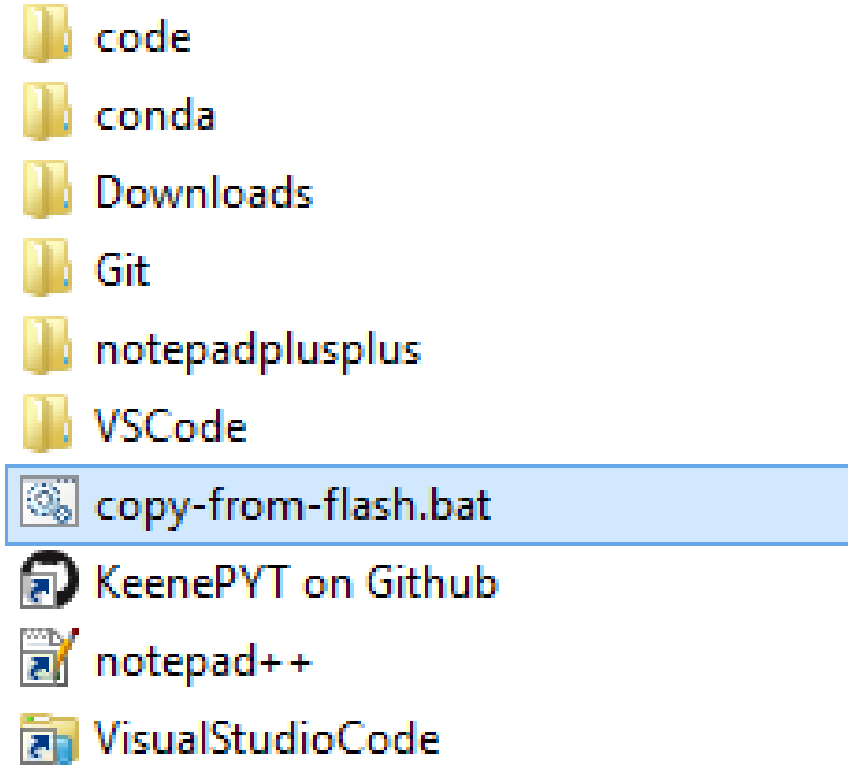
Help us get started.

1. Get a flash drive.
2. Run **copy-from-flash.bat**.

This will create **C:\Projects\NEARC**.

The same code is available at **https://github.com/jswise/keenepyt**.



Distribute Your Python Tools to ArcGIS Pro Users

Jason Wise, Earth Data Scientist, Terracon

Terracon

Distribute Your Python Tools to ArcGIS Pro Users

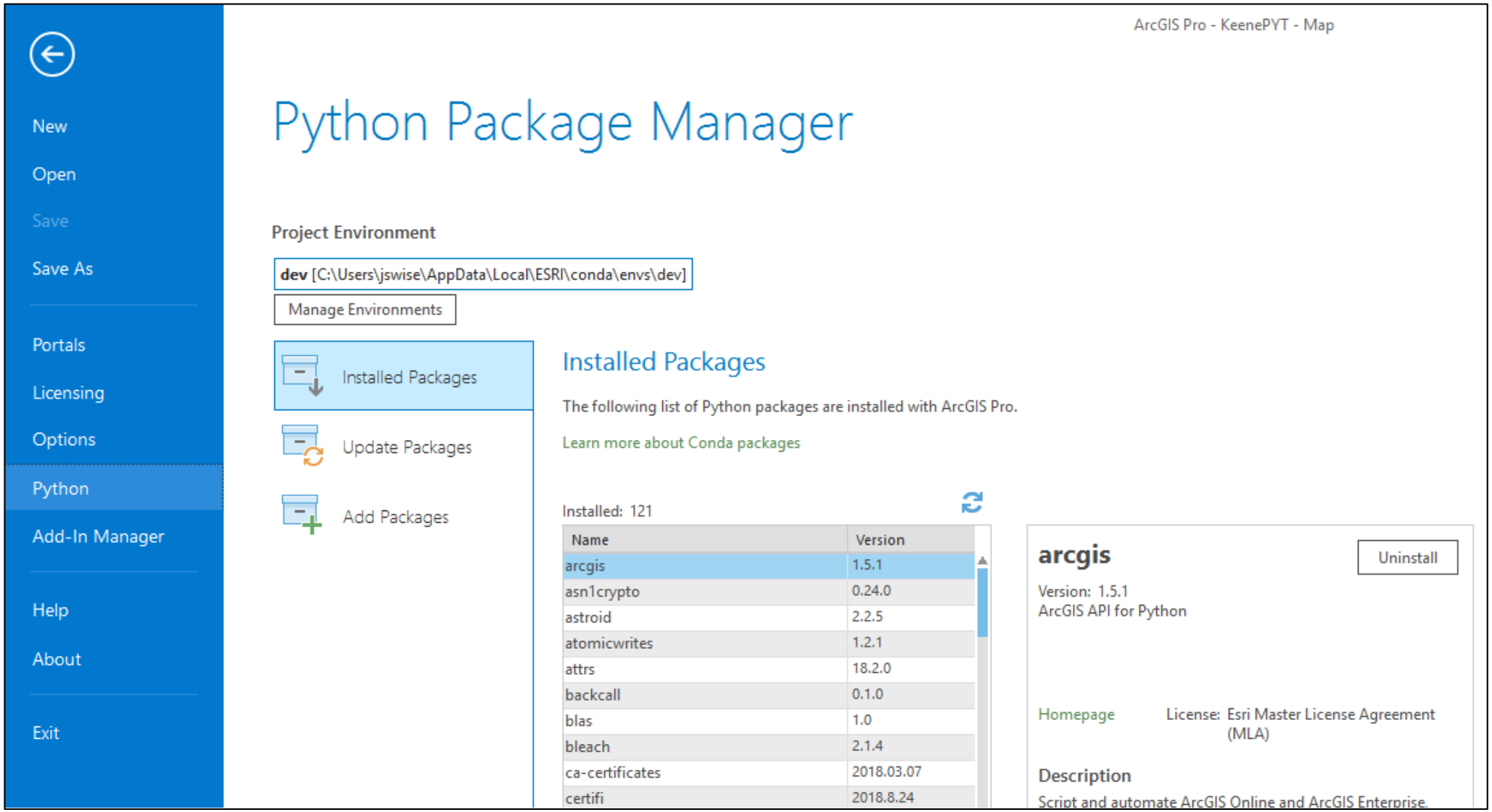
Jason Wise, Earth Data Scientist

Terracon

Northeast Arc User Group (NEARC)
Spring 2019

The Python Package Manager

It manages “environments” and “packages.” We’ll talk about what those are.

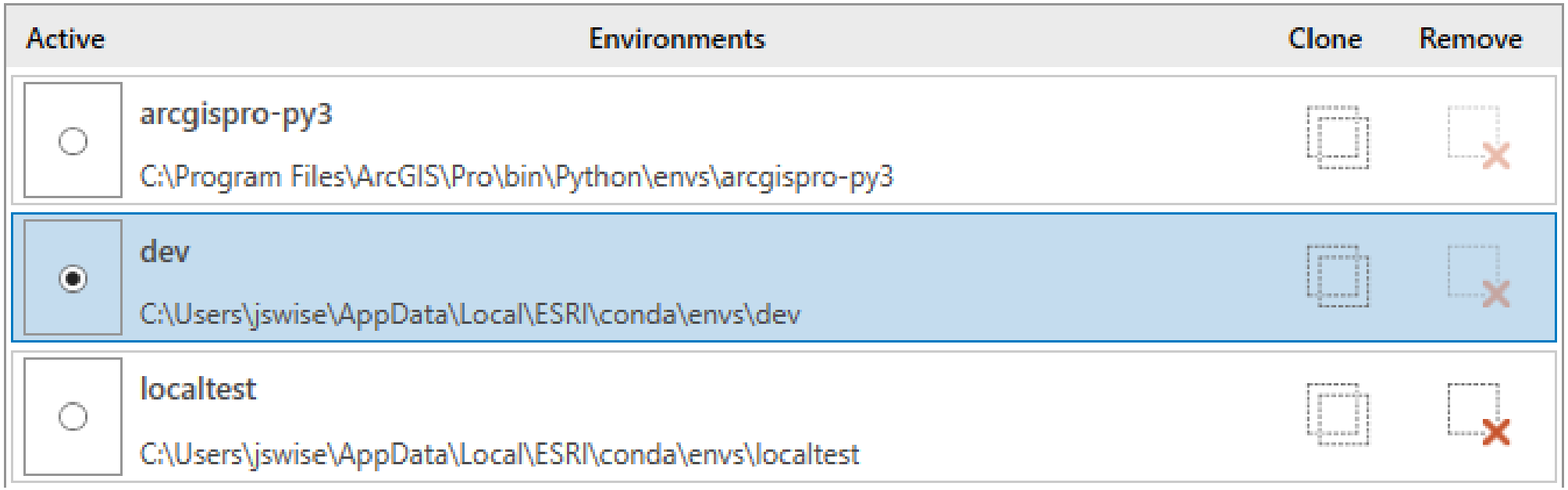


Terracon

Create dev & test environments.

In **C:\Projects\NEARC\code\keenepyt\batch**, run **create-dev-envs.bat.**

This will clone your default Python environment to two new environments, *dev* and *localtest*. It takes a while to run.



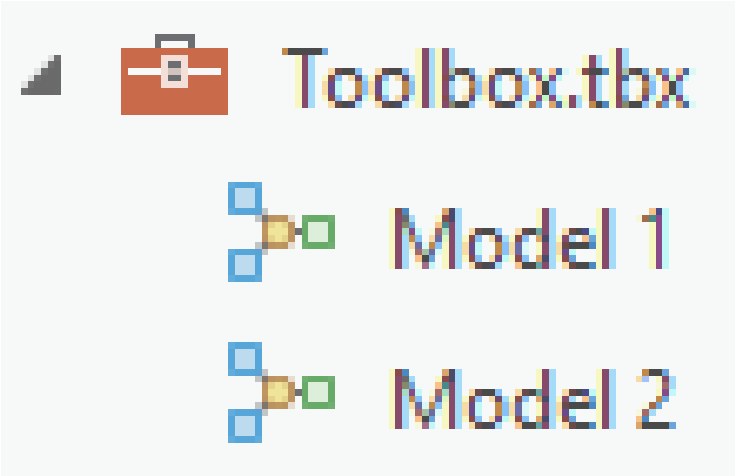
The goals:

1. Give a geoprocessing toolbox to other users.
2. Easily update the toolbox.

With traditional methods (e.g. passing out .TBX files and Python scripts), updating is the hard part.

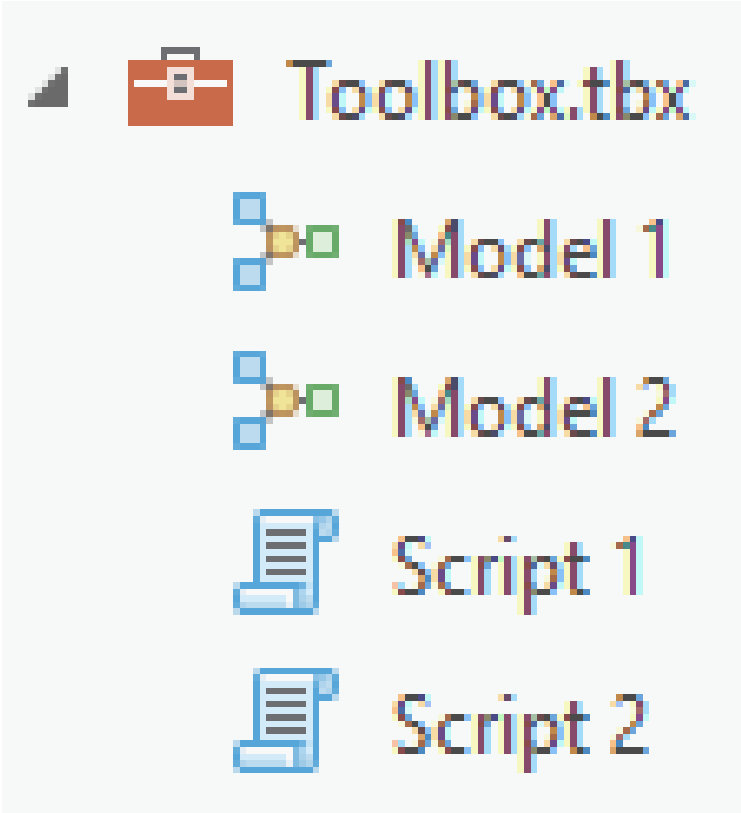
The Toolbox Menagerie

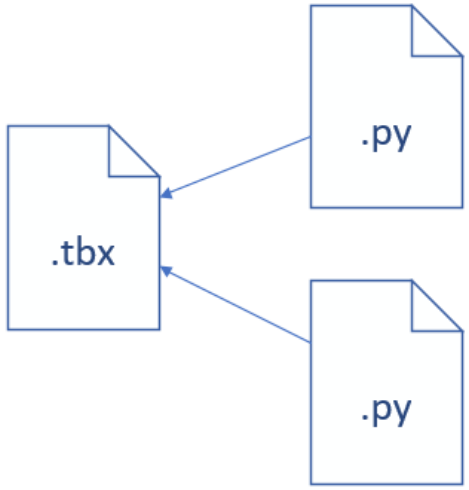
A traditional toolbox is a binary file.



The Toolbox Menagerie

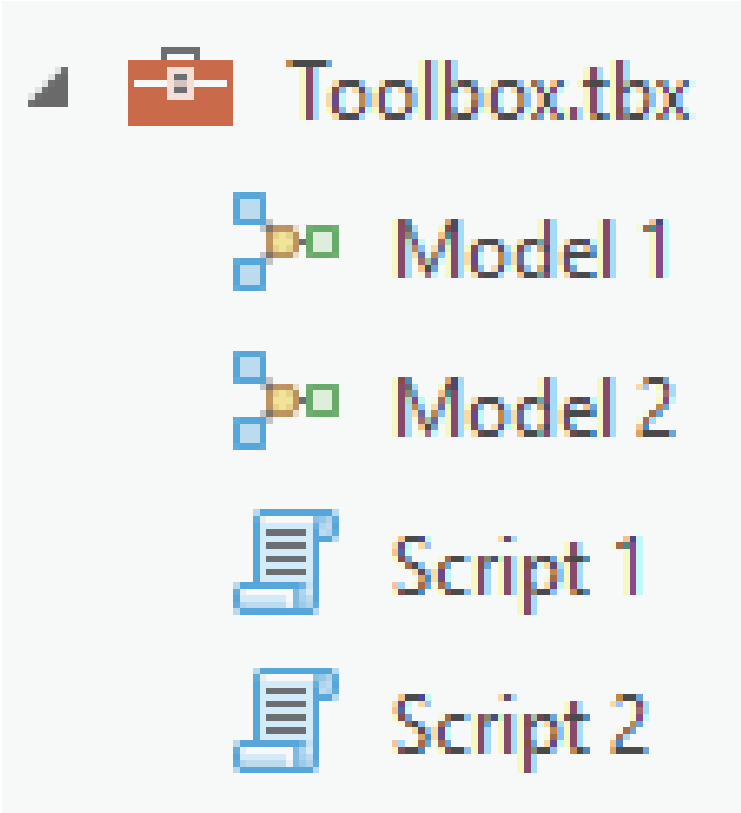
Each script is a Python module (file).

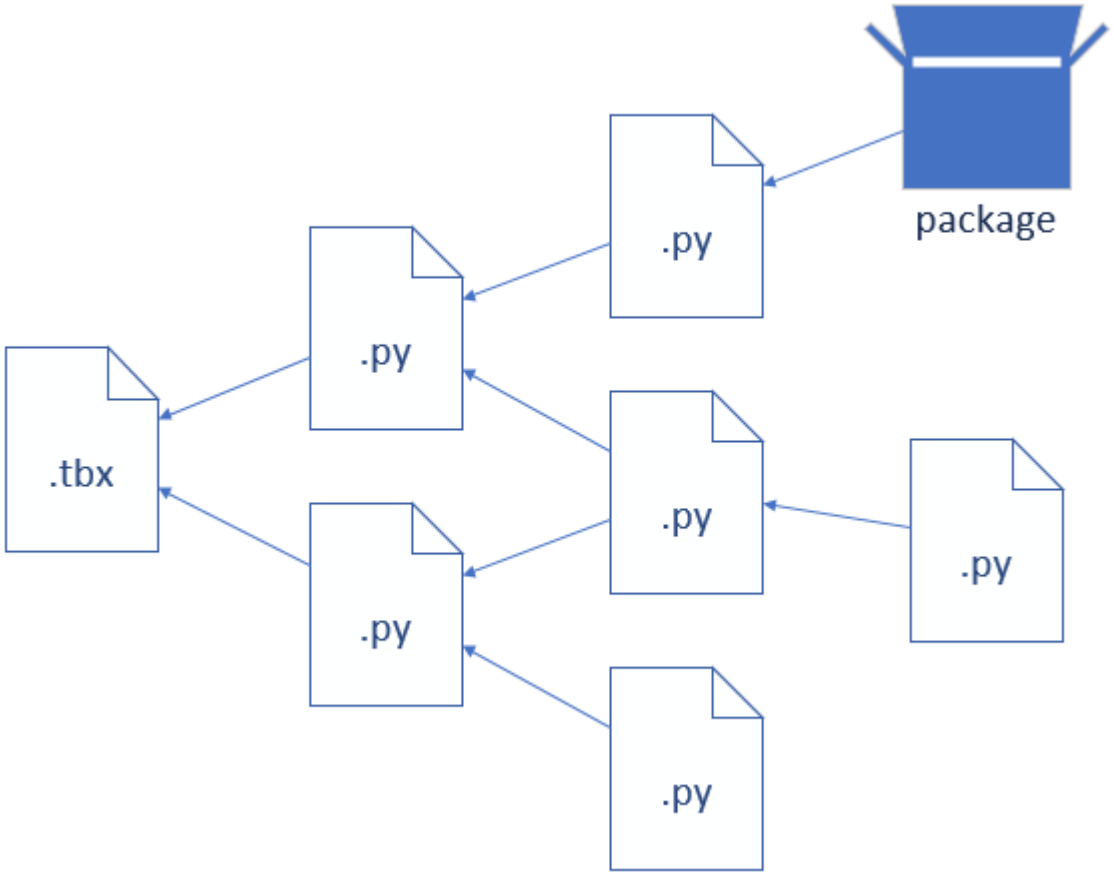




The Toolbox Menagerie

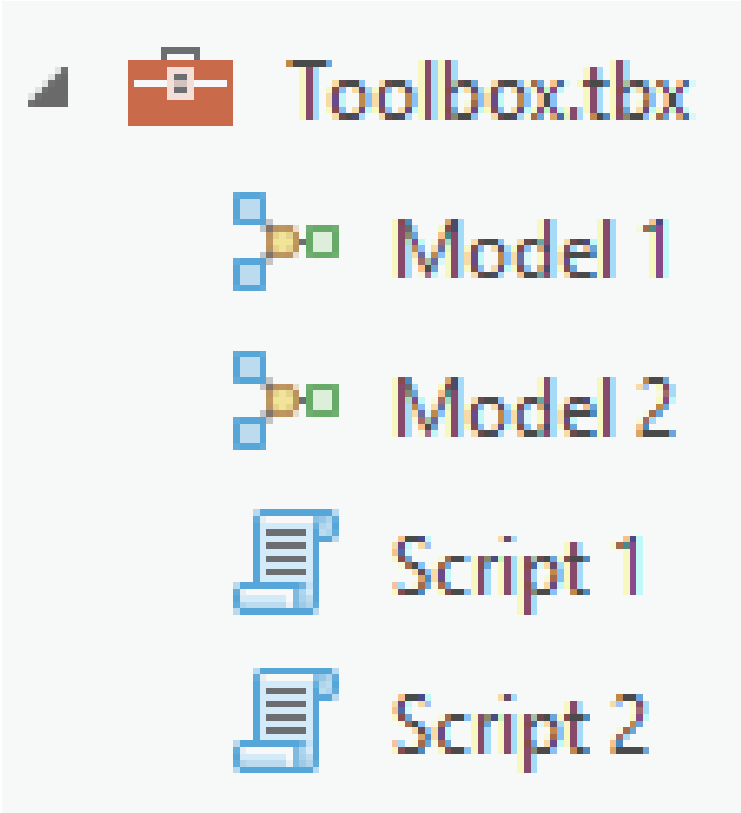
Each Python module can import other modules & packages.

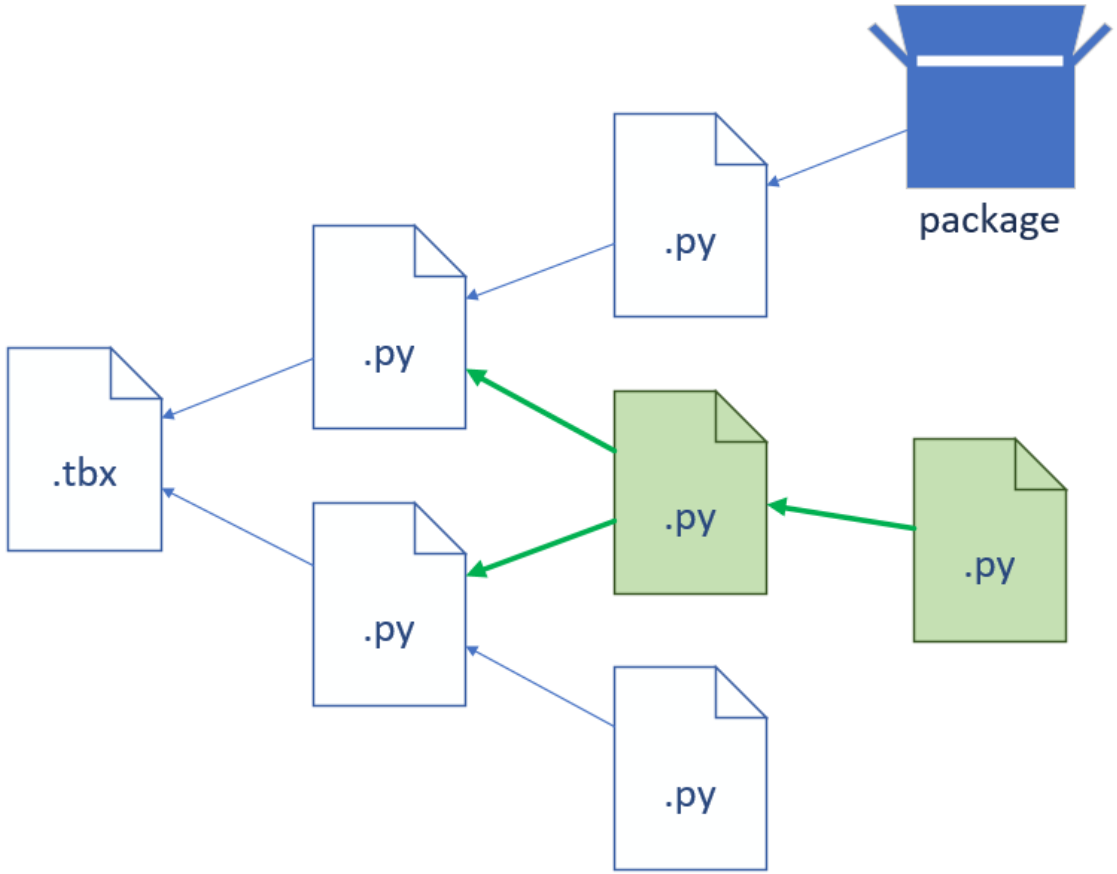




The Toolbox Menagerie

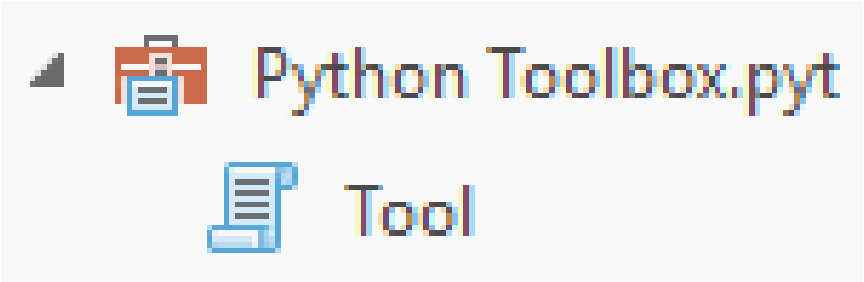
Reusing code is good!





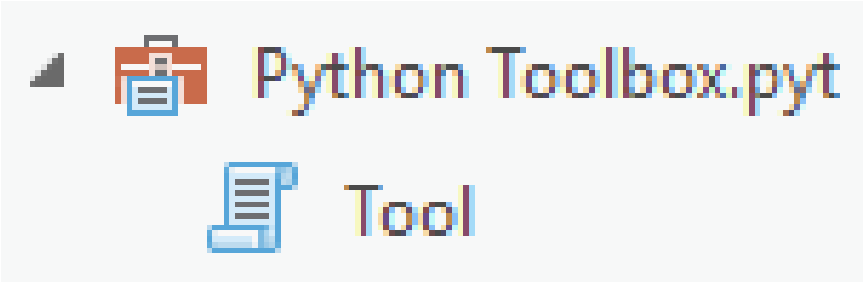
The Toolbox Menagerie

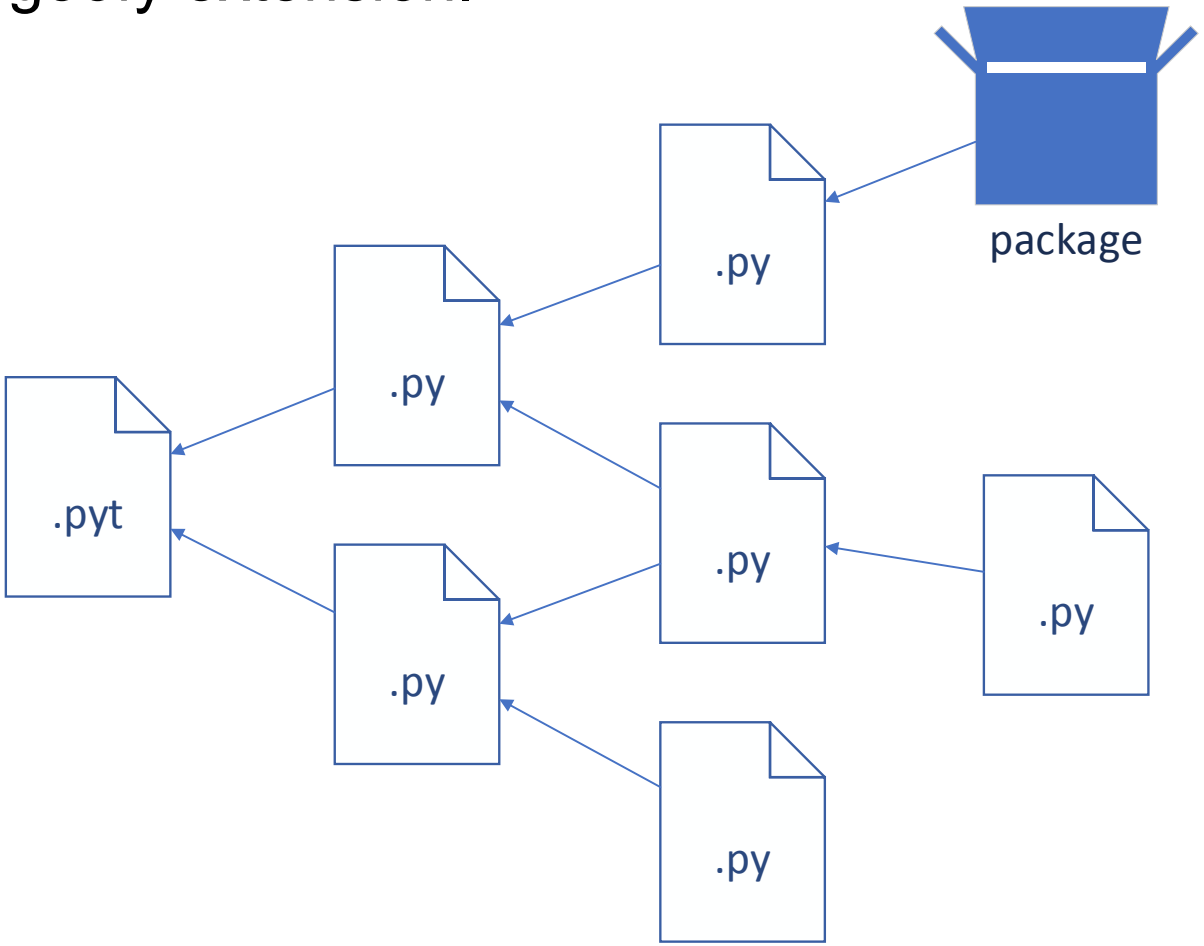
A Python toolbox is a Python file with a goofy extension.
It can be self-contained…



The Toolbox Menagerie

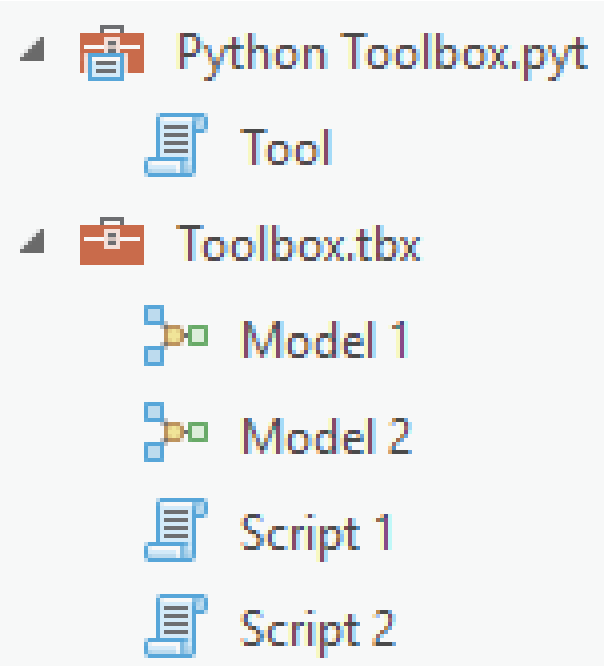
A Python toolbox is a Python file with a goofy extension.
It can be self-contained…
…or import other modules & packages.

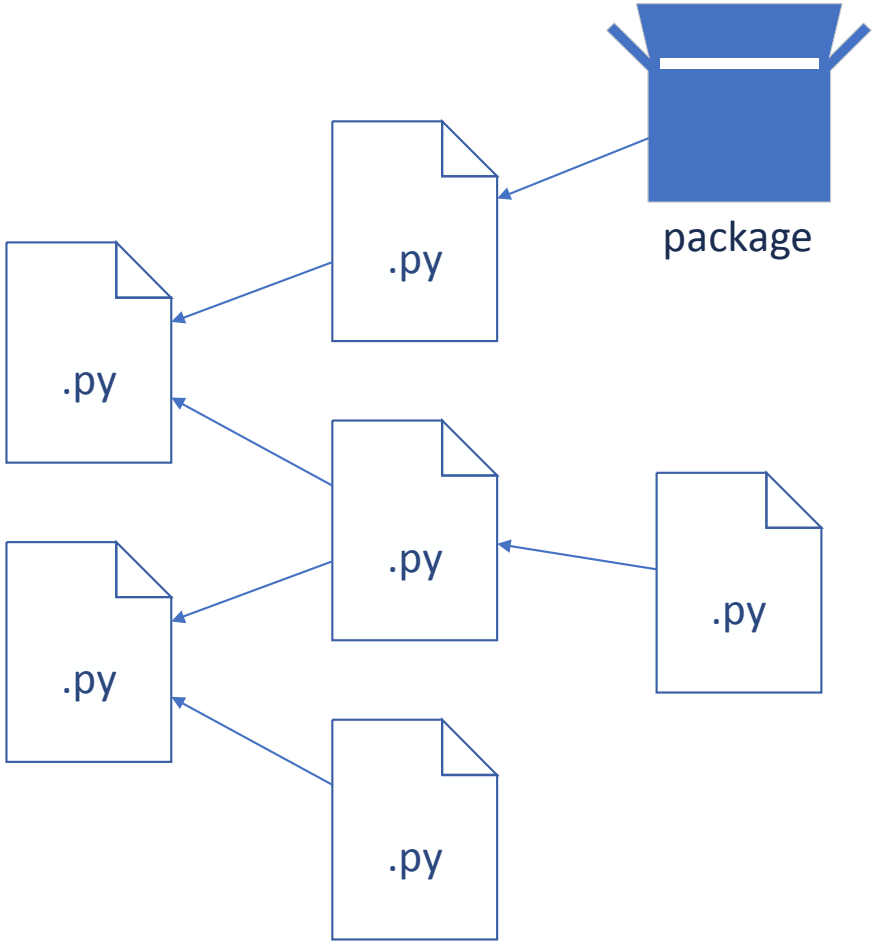




The Toolbox Menagerie

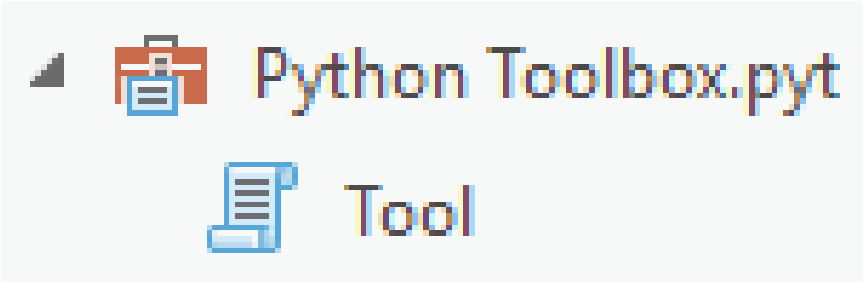
Distributing these to users can be a pain.
Updating them is worse.

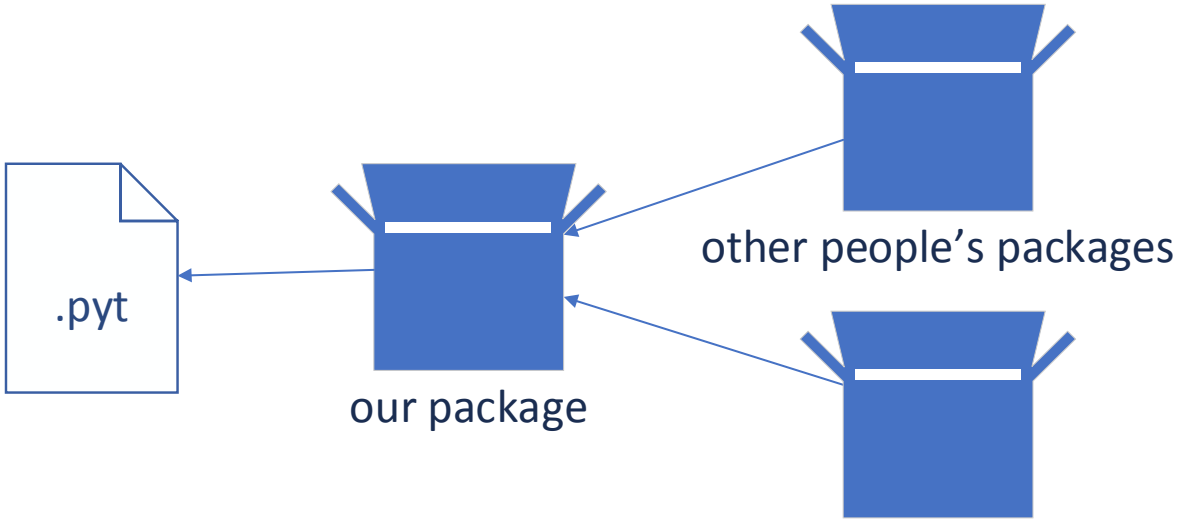




Our approach

We'll put all our code in a package, then import it into a Python toolbox.





Python packages in Pro

ArcGIS Pro comes with lots of handy Python packages that you can import into your code…

Python

```
import pandas
```

Python packages in Pro

ArcGIS Pro comes with lots of handy Python packages that you can import into your code…
…but not all of them.

Python

```
import pandas
import pyodbc
Traceback (most recent call last):
  File "<string>", line 1, in <module>
ModuleNotFoundError: No module named 'pyodbc'
```

Python packages in Pro

The Python Package Manager makes it easy to get more of them.

Add Packages

Python packages let you do more with ArcGIS Pro. The list below includes available Python packages that can be optionally installed.

Search

Name	Versions
pynacl	1.3.0
pyodbc	4.0.26
pyopengl	3.1.1.1
pyopengl-accelerate	3.1.3.1
pypandoc	1.4
pyprof2calltree	1.4.4
pyproj	2.1.1
pyqtgraph	0.10.0
pyquery	1.4.0
pyramid	1.10.2
pyramid_debugtoolbar	4.4
pyramid_jinja2	2.7
pyramid_mako	1.0.2
pyramid_tm	2.2.1
pyreadline	2.1

pyodbc

Install

Version: 4.0.26
DB API Module for ODBC

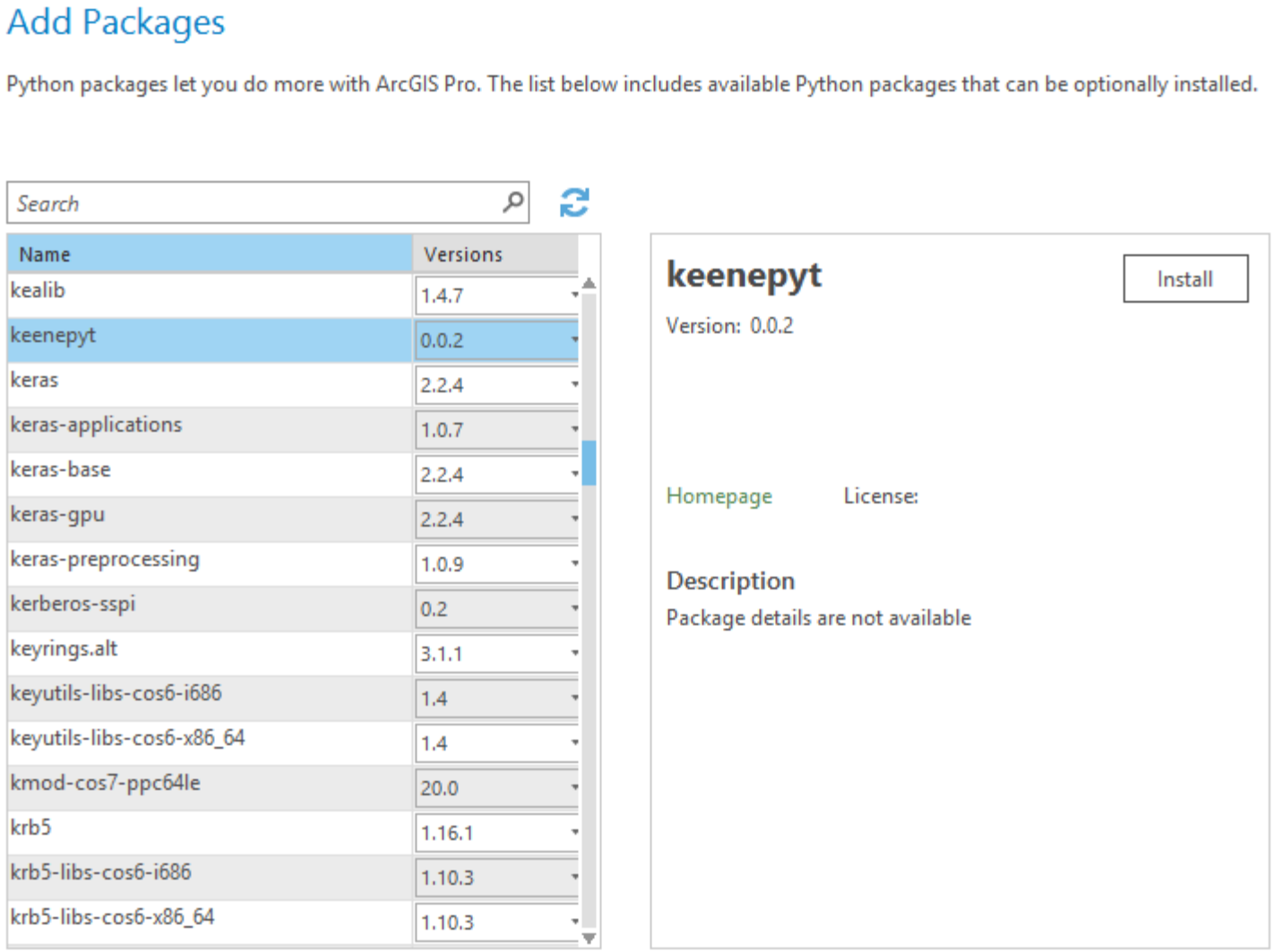
Homepage License: MIT

Description

pyodbc is a Python DB API 2 module for ODBC. This project provides an up-to-date, convenient interface to ODBC using native data types like datetime and decimal.

Python packages in Pro

How about making your own package?



Our approach

Updating will be easy.

Update Packages

The following list of Python packages are installed with ArcGIS Pro and have recent updates.

Updates: 84

Update All

Name	Version
jupyter_console	5.2.0
keenepyt	0.0.3
keyring	13.2.1
kiwisolver	1.0.1
libpng	1.6.34
markupsafe	1.0
matplotlib	2.2.3
mistune	0.8.3
mkl	2018.0.3
mkl_fft	1.0.4
mkl_random	1.0.1
more-itertools	4.3.0
mpmath	1.0.0
nbconvert	5.3.1
netcdf4	1.4.1
notebook	5.6.0
numexpr	2.6.8
numpy	1.15.1
numpy-base	1.15.1

keenepyt

Update

Version: 0.0.3

Homepage License:

Description

Package details are not available

Our approach

Esri doesn’t want us to modify the default environment, so we cloned it.

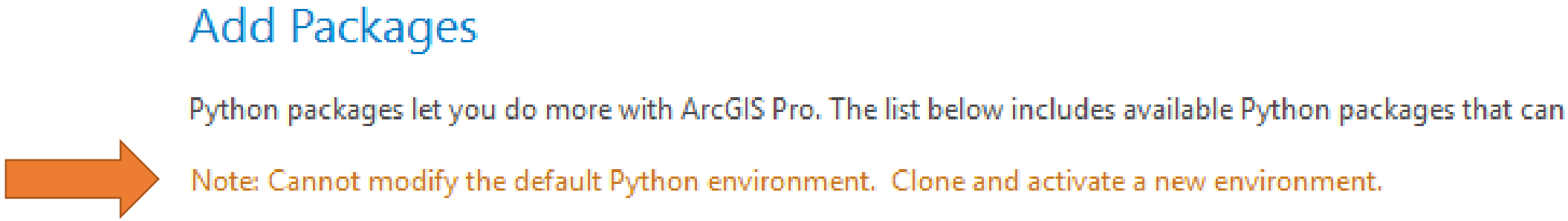
Add Packages

Python packages let you do more with ArcGIS Pro. The list below includes available Python packages that can

Note: Cannot modify the default Python environment. Clone and activate a new environment.

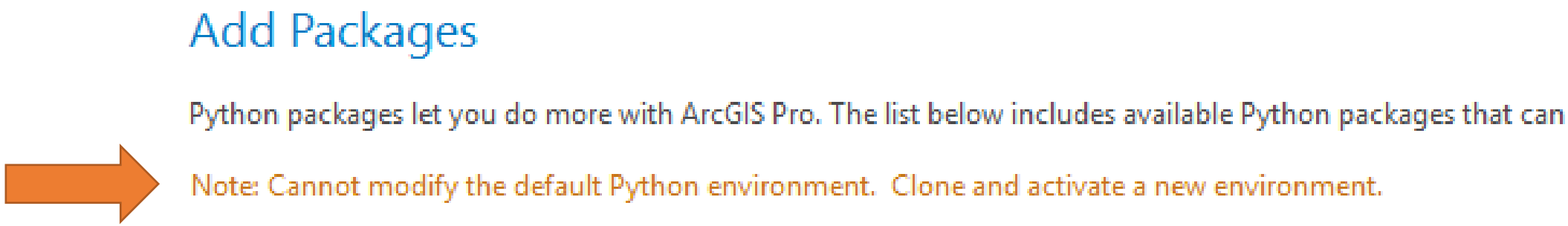
Our approach

Esri doesn’t want us to modify the default environment, so we cloned it.
Think of an “environment” as a Python installation.



Our approach

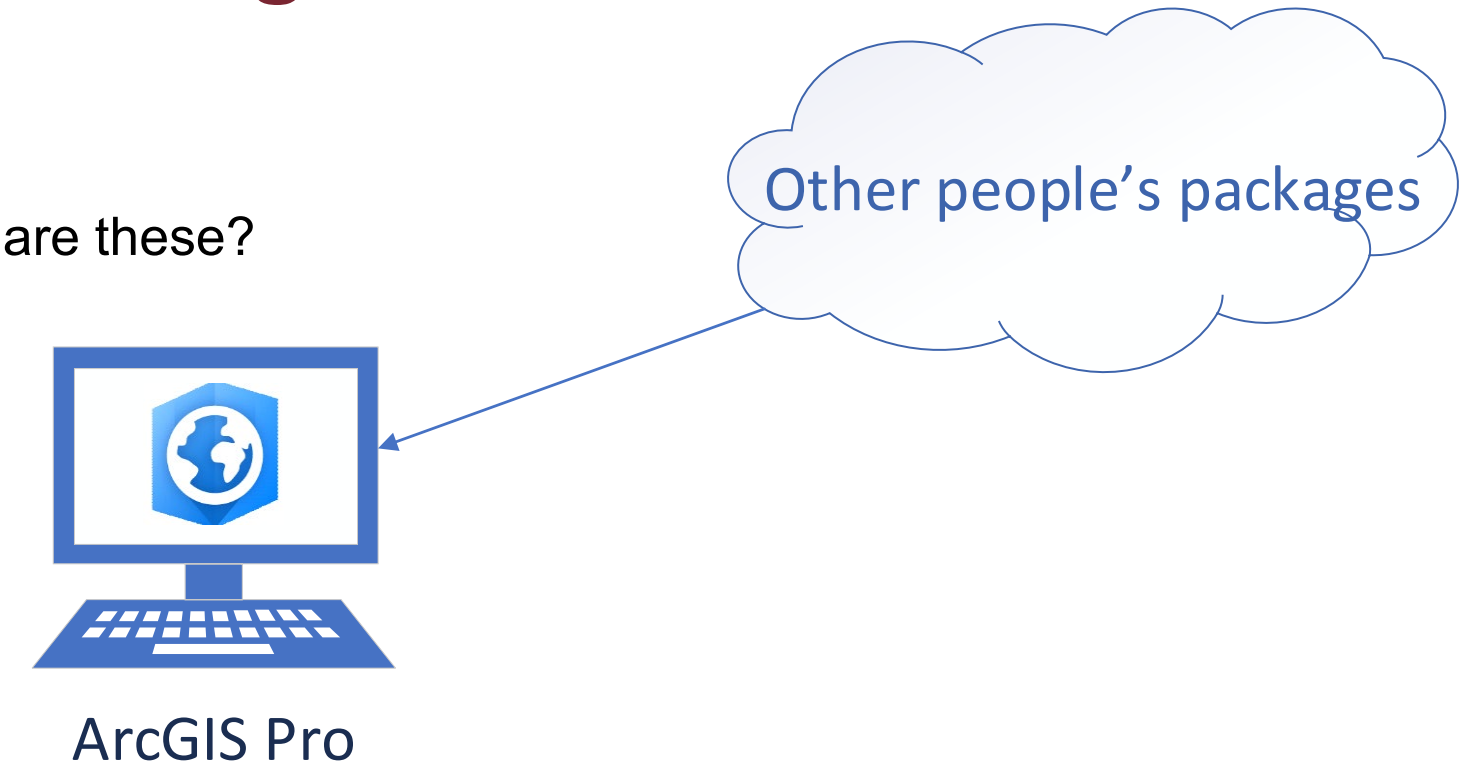
Esri doesn’t want us to modify the default environment, so we cloned it.



Note: We could get around this restriction by doing it outside of Pro, but we won’t.

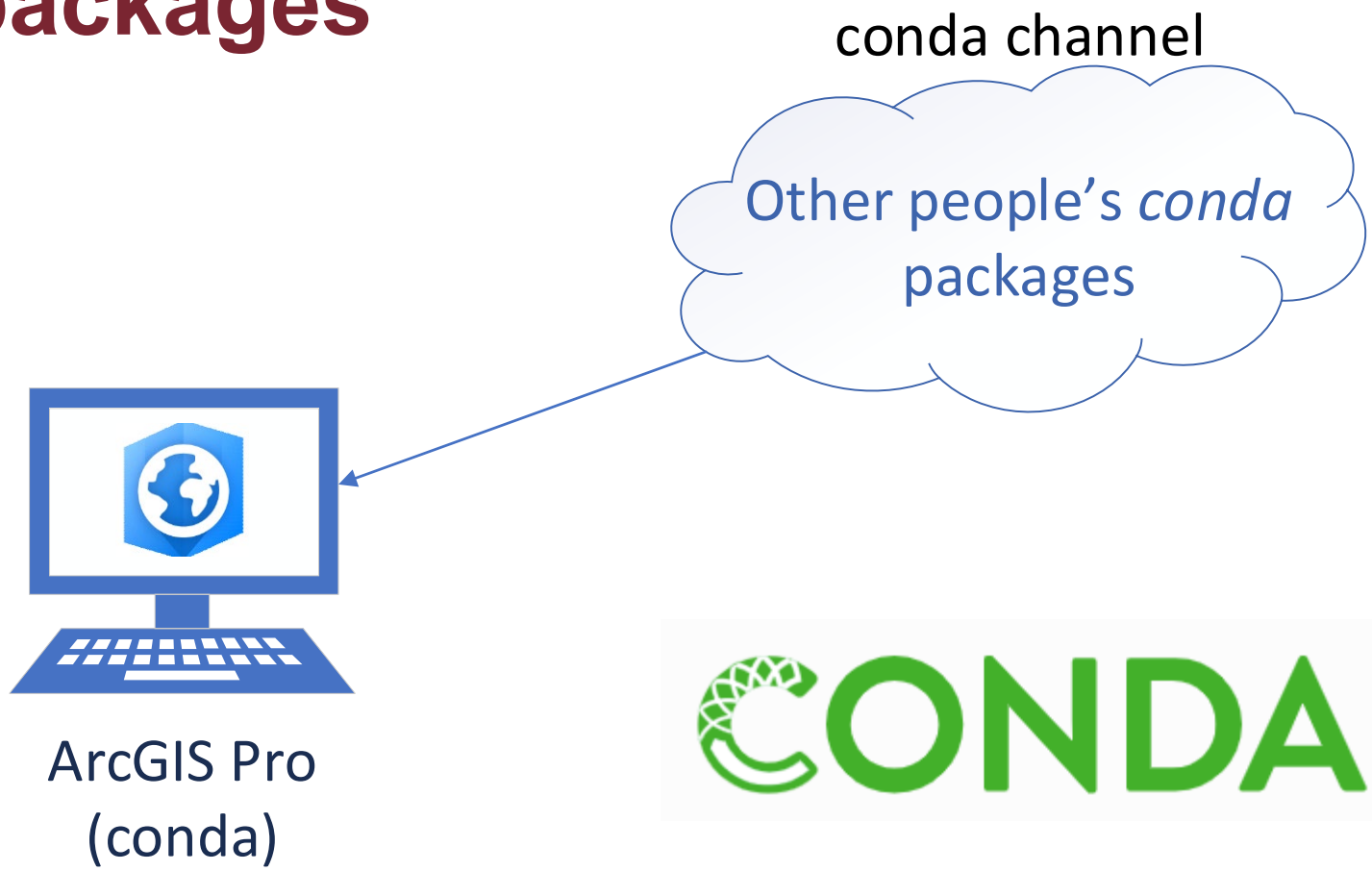
How Pro gets packages

What kind of “packages” are these?



How Pro gets packages

They're conda packages.



How Pro gets packages

They're conda packages.

Conda is open-source software for managing environments and packages.

ArcGIS Pro (conda)

conda channel

Other people's *conda* packages

How Pro gets packages

They're conda packages.

Conda is open-source software for managing environments and packages.

The "c" is lowercase.

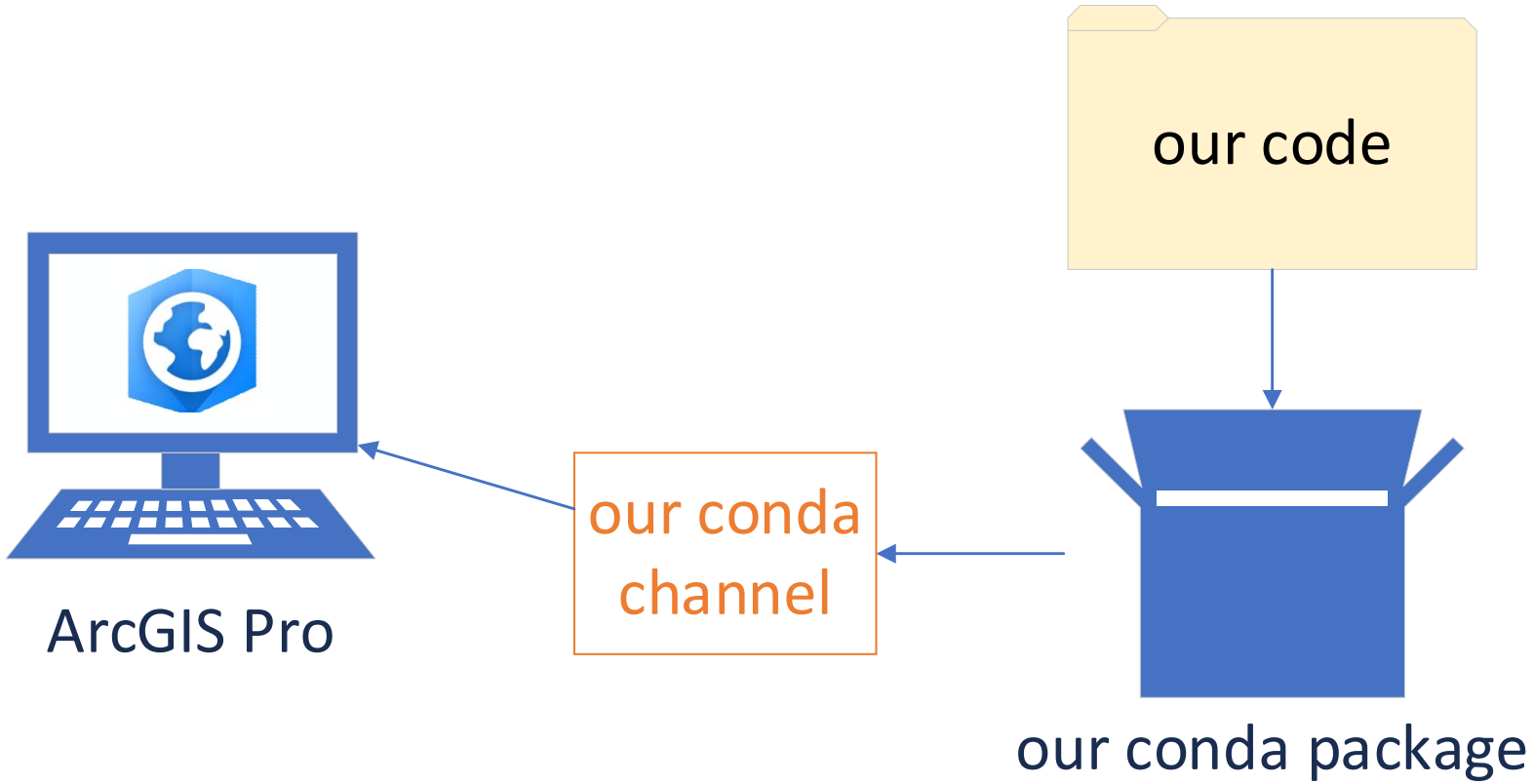
ArcGIS Pro (conda)

conda channel

Other people's *conda* packages

How Pro gets packages

We'll make our own conda package and channel.



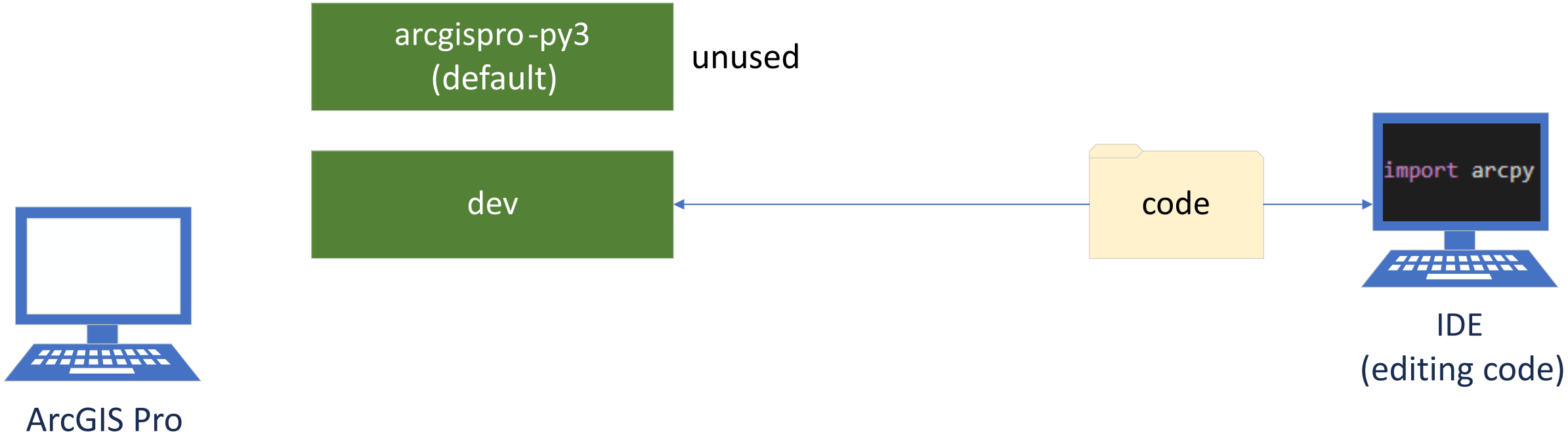
Our environments

arcgispro-py3
(default)

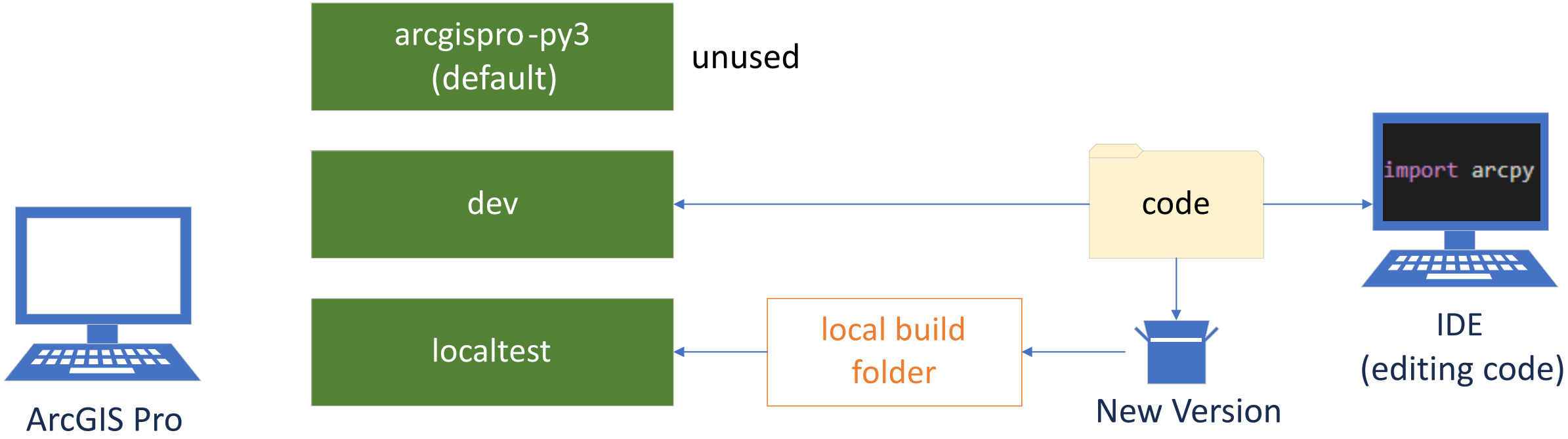
unused

ArcGIS Pro

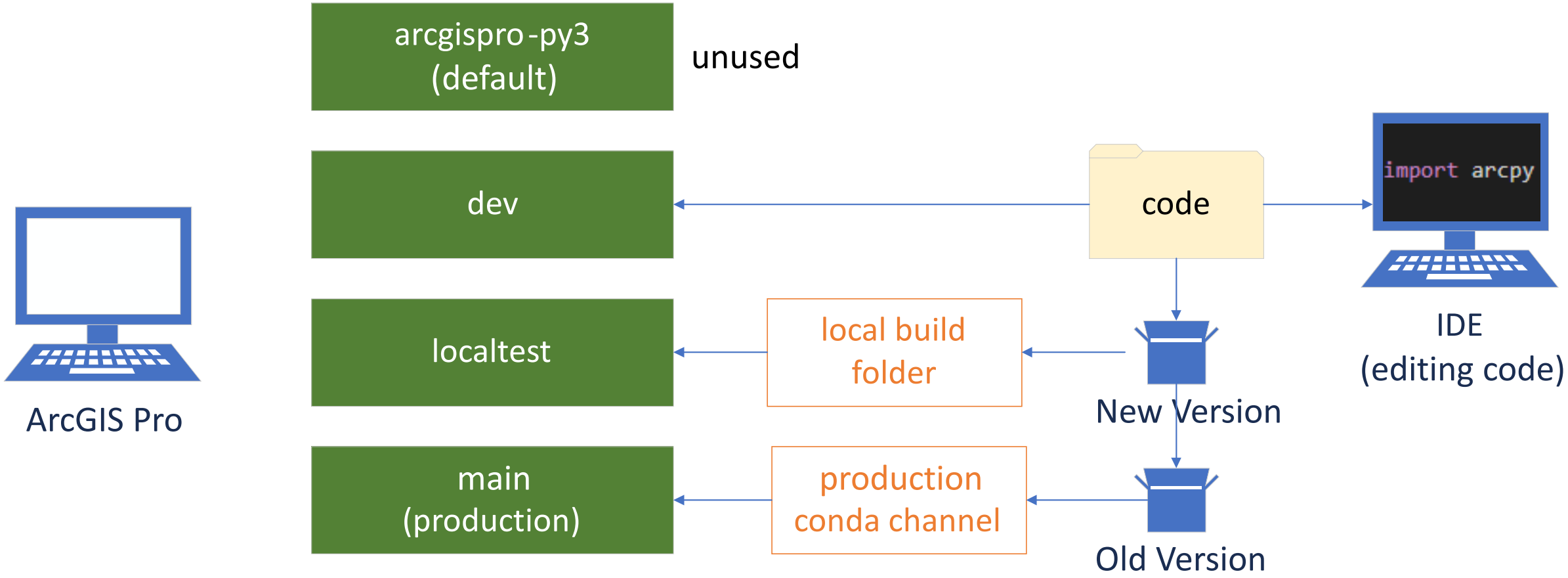
Our environments



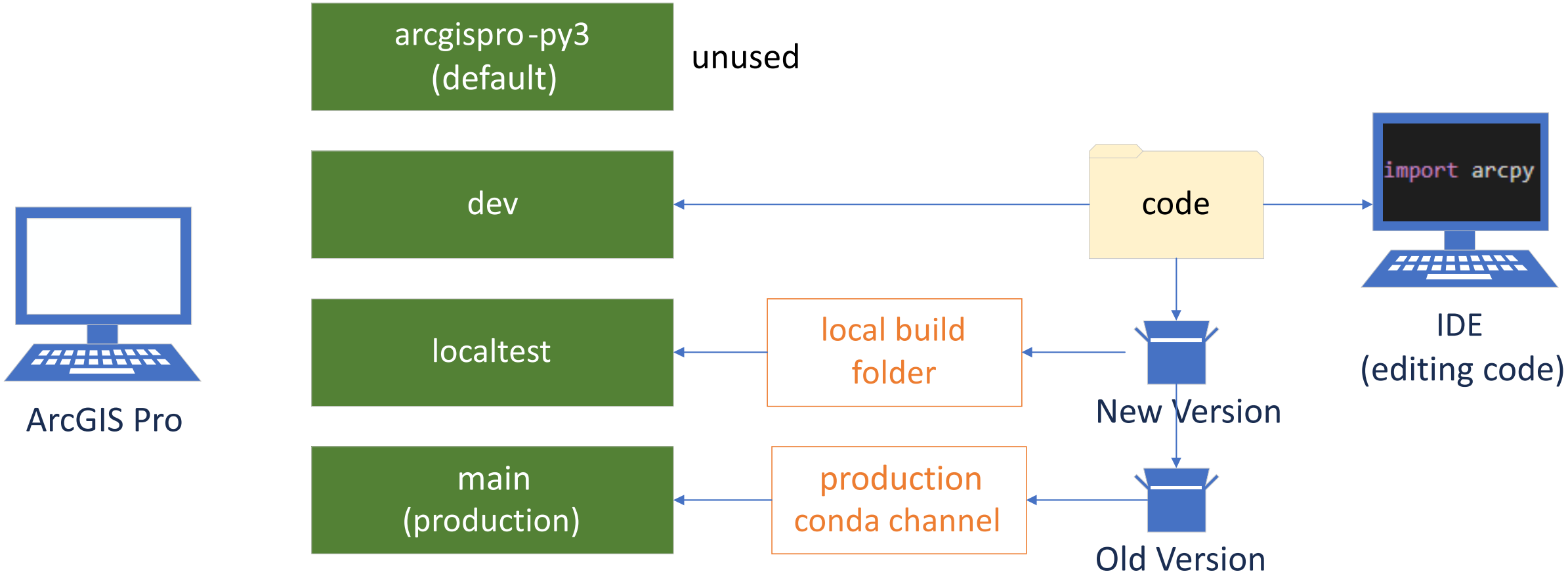
Our environments



Our environments



Our environments

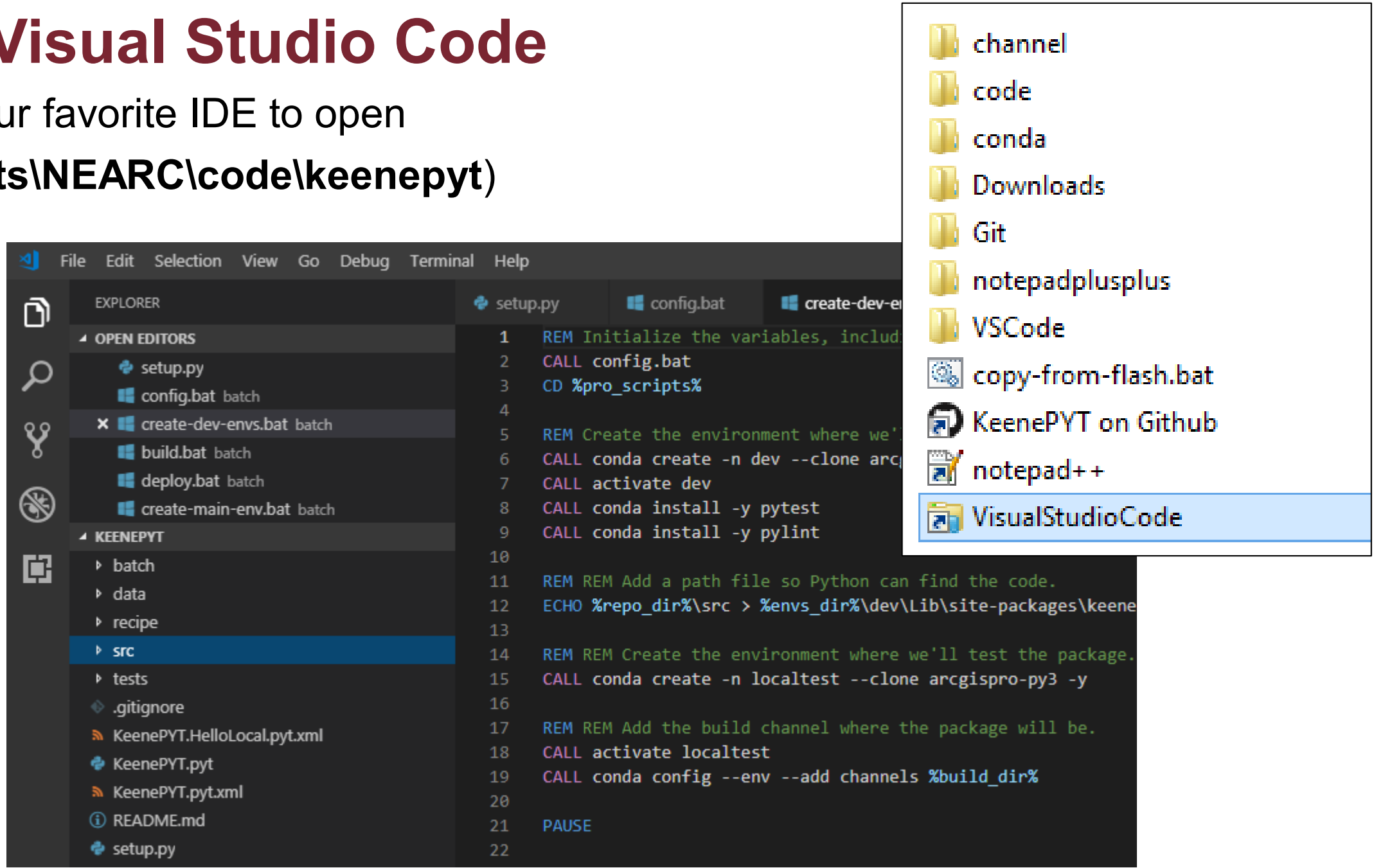


+build environment (separate conda installation)

Open Visual Studio Code

(or use your favorite IDE to open

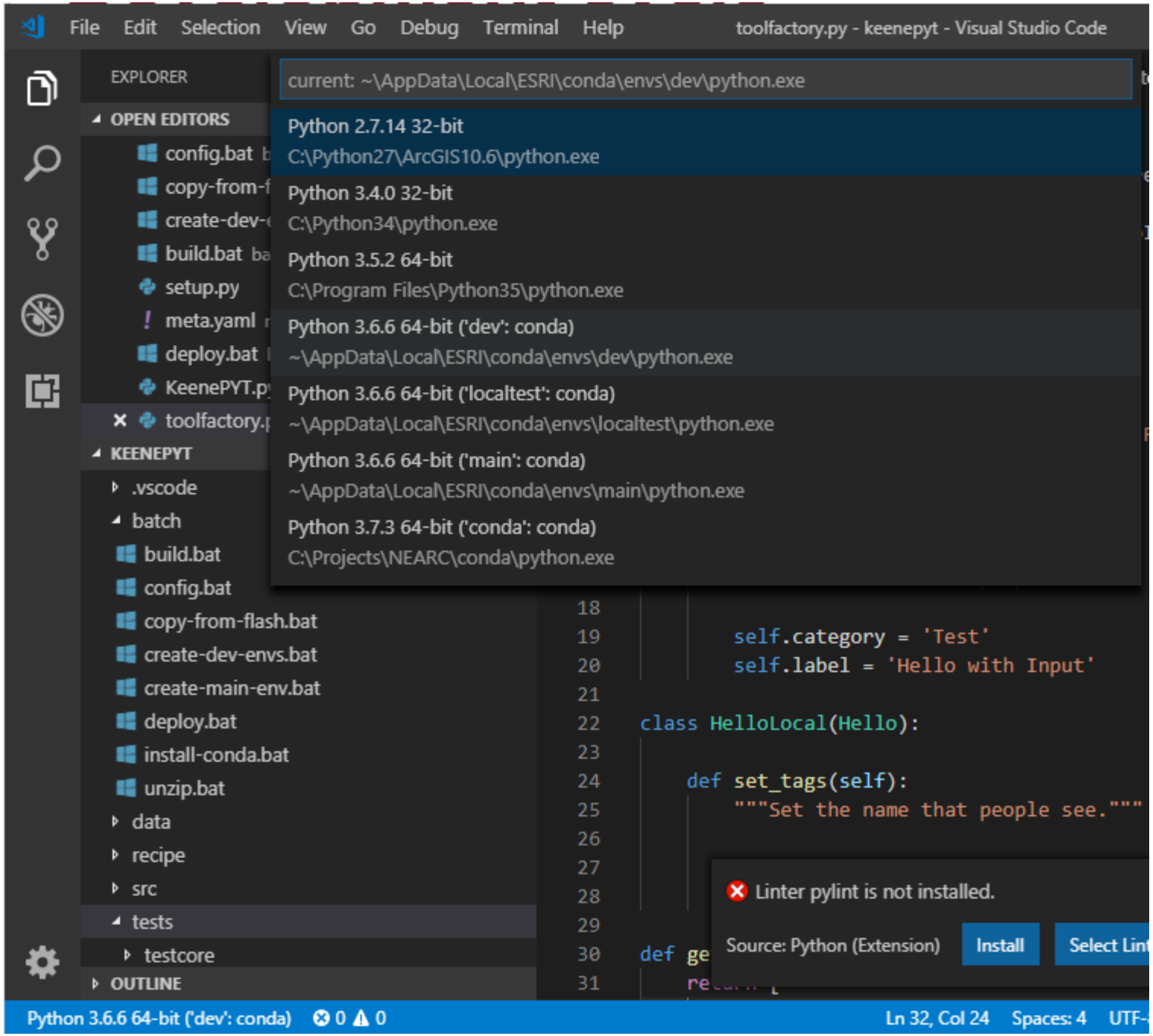
C:\Projects\NEARC\code\keenepyt)



Terracon

Set the Python interpreter.

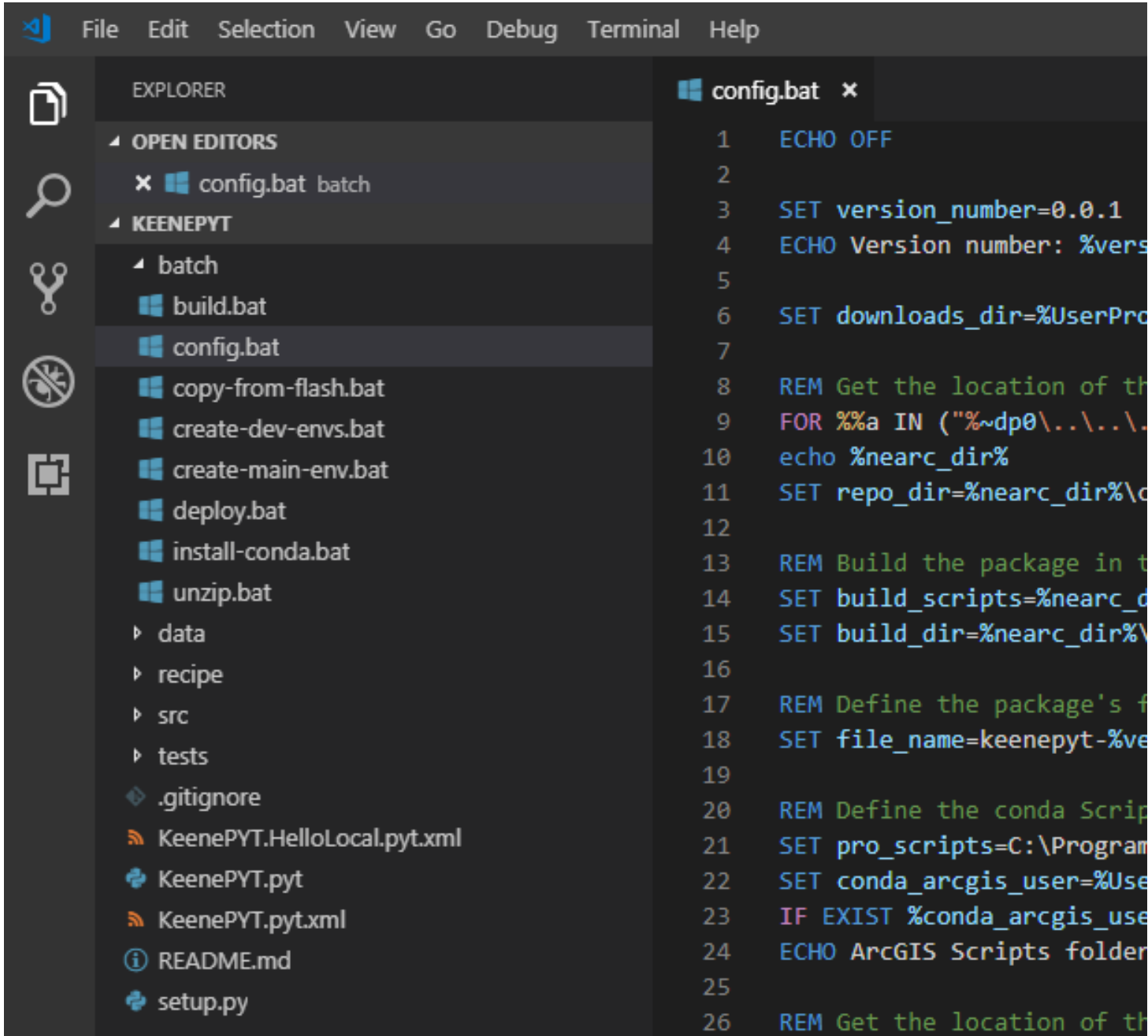
Click on “Python” in the lower-left corner, then choose “dev.”



Terracon

View the batch files.

In your IDE, open **batch\config.bat**.

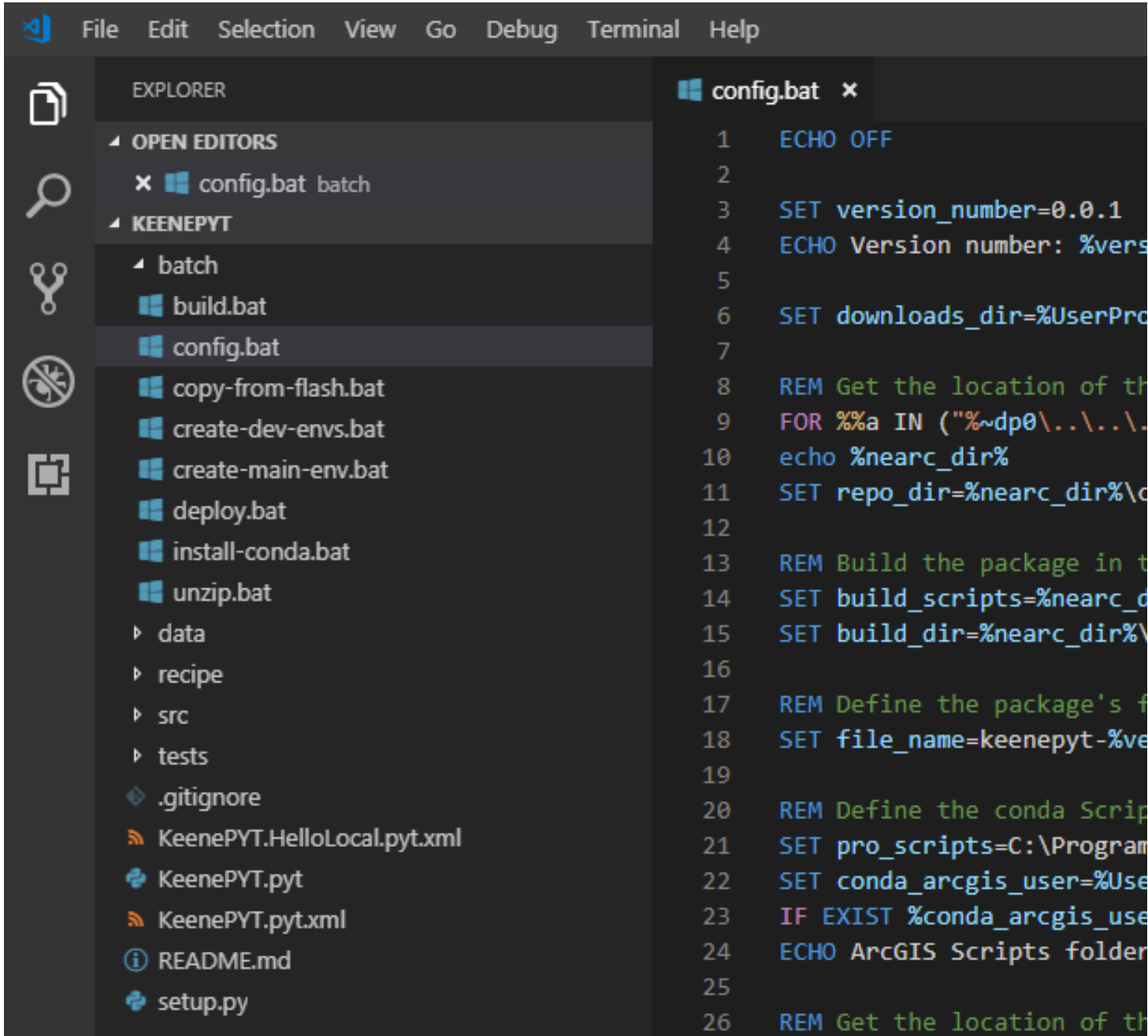


Terracon

View the batch files.

In your IDE, open **batch\config.bat**.

This batch file sets a bunch of variables that other batch files will use, including the version number of our package.

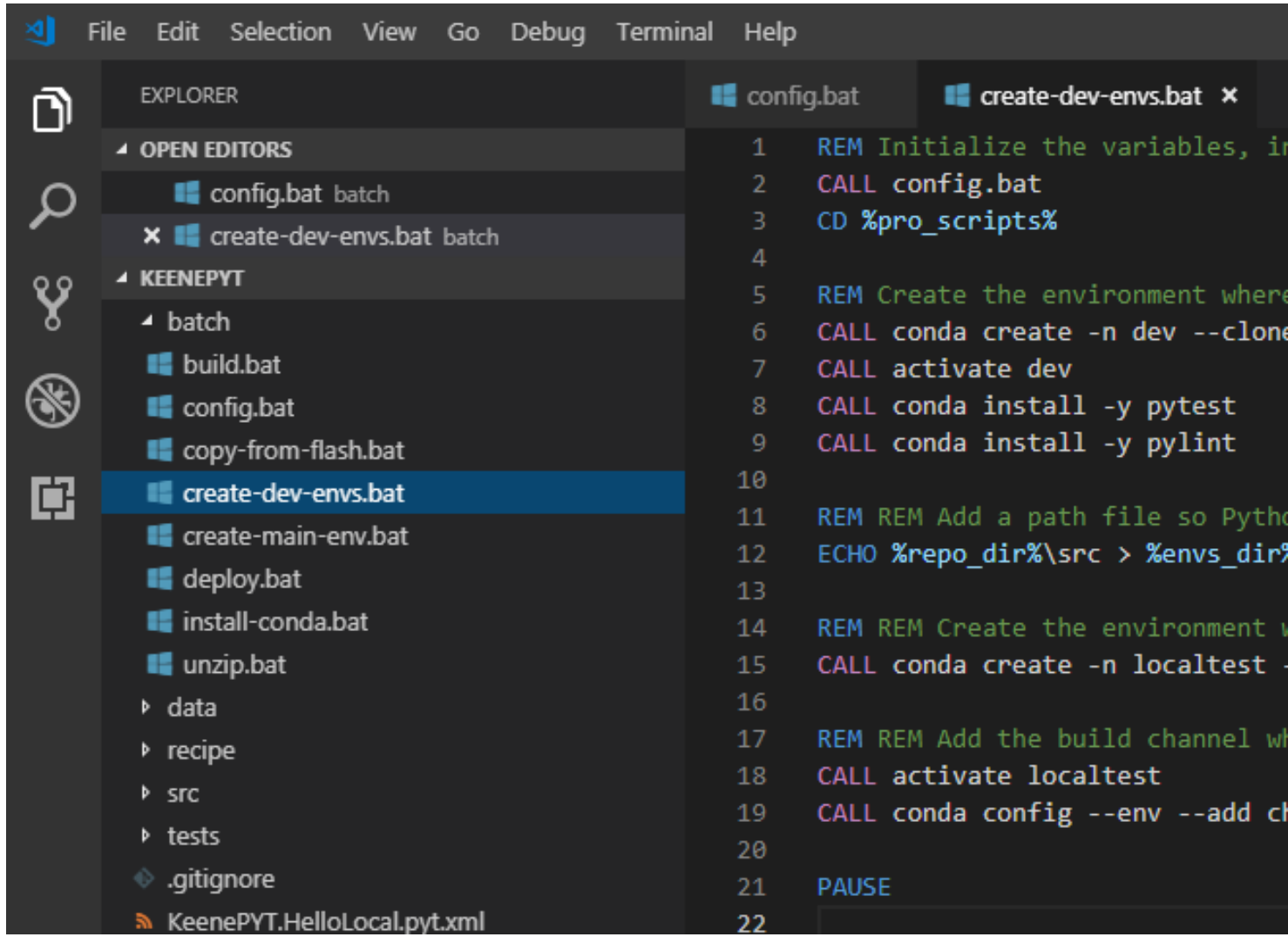


Terracon

View the batch files.

Open **create-dev-envs.bat**.

You ran this one earlier.



Terracon

View the batch files.

Open **create-dev-envs.bat**.

Note this line:

```
REM Add a path file so Python can find the code.
%repo_dir%\src > %envs_dir%\dev\Lib\site-packages\keenepyt.pth
```

It creates a path file (.pth) that contains the path to your code.

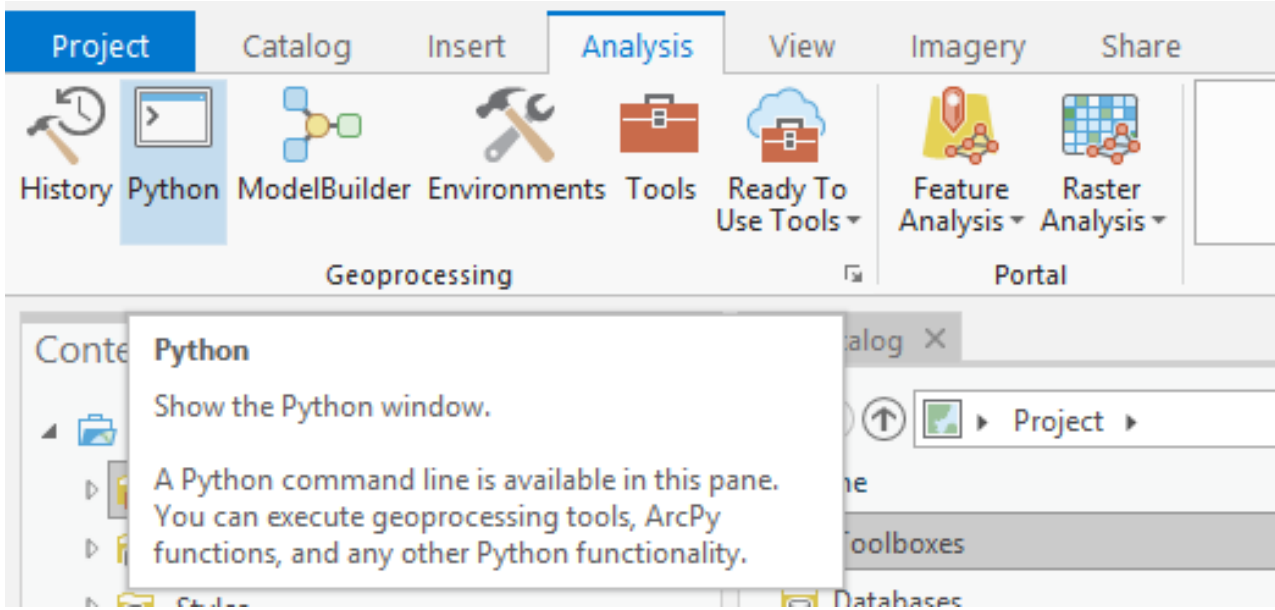
Activate the dev environment.

Open ArcGIS Pro, and use the Python Package Manager to activate dev.

Active	Environments	Clone	Remove
○	arcgispro-py3 C:\Program Files\ArcGIS\Pro\bin\Python\envs\arcgispro-py3		
◉	dev C:\Users\jswise\AppData\Local\ESRI\conda\envs\dev		
○	localtest C:\Users\jswise\AppData\Local\ESRI\conda\envs\localtest		

Try importing keenepyt.

This should work in the dev environment.

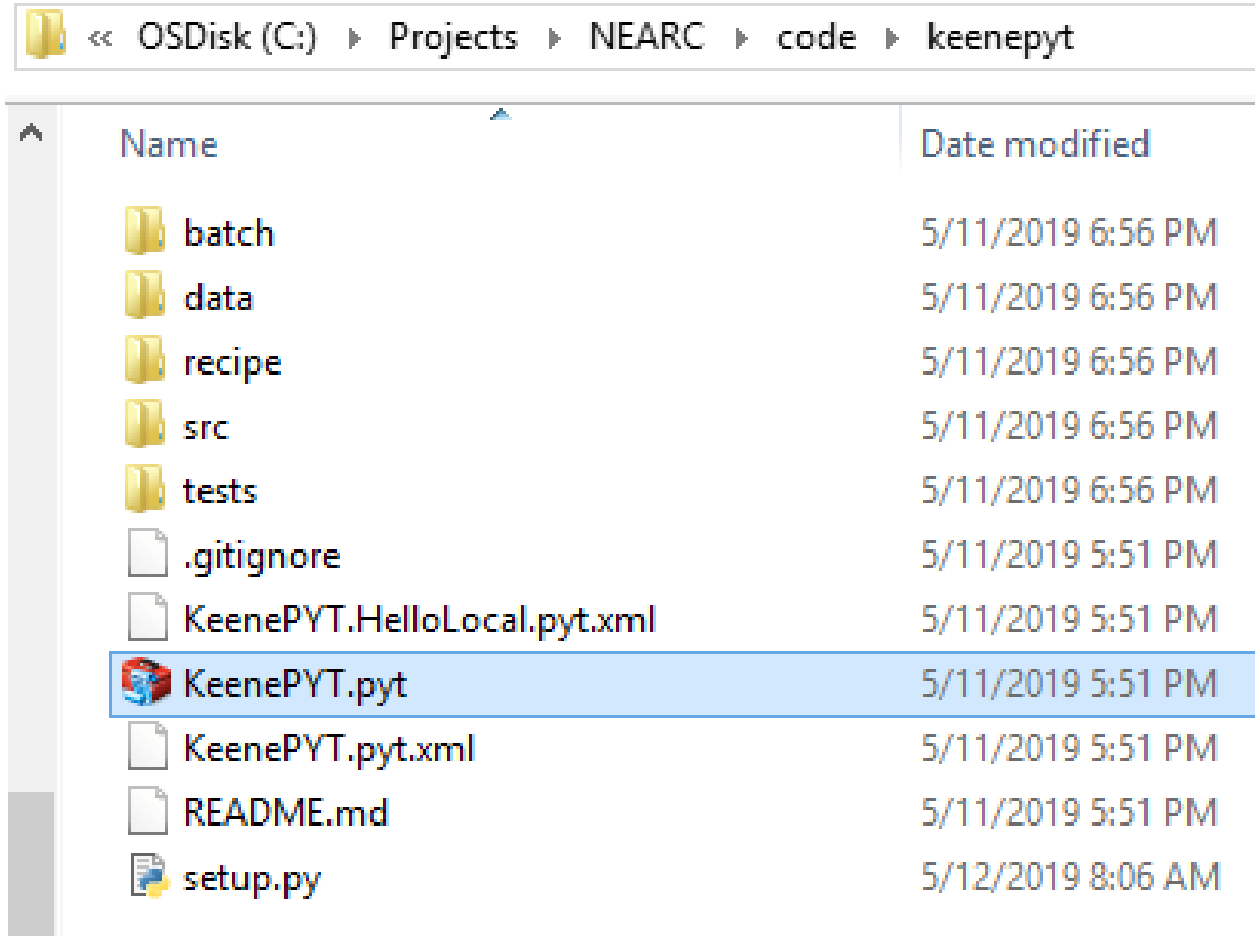



```
import keenepyt
```

Pretend to distribute the toolbox.

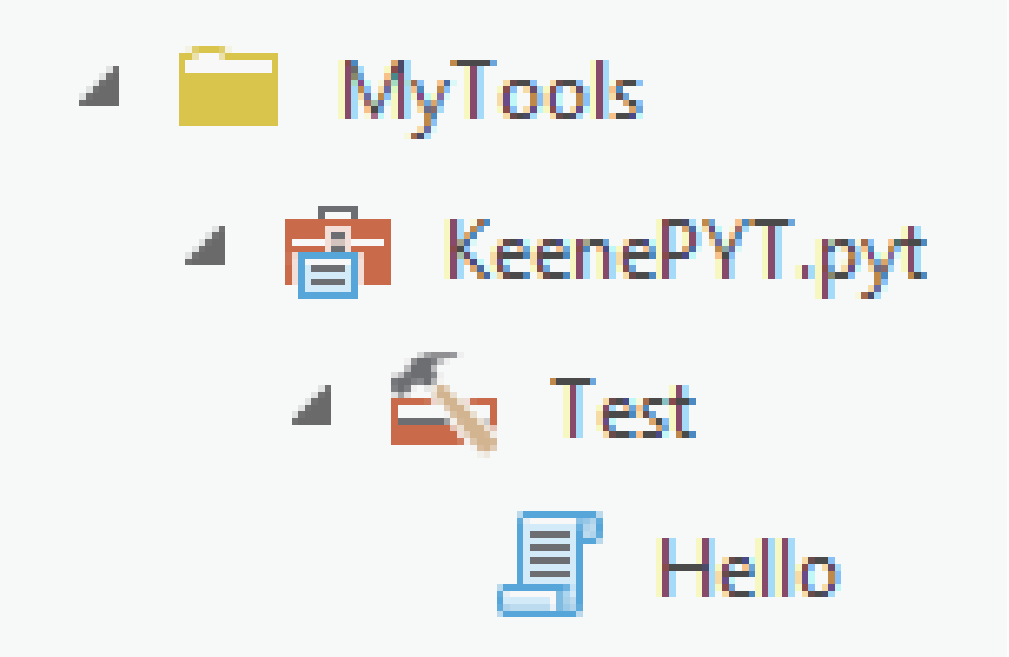
Find KeenePYT.pyt, and copy it to some other place.

This simulates giving your toolbox to somebody.



Open the toolbox.

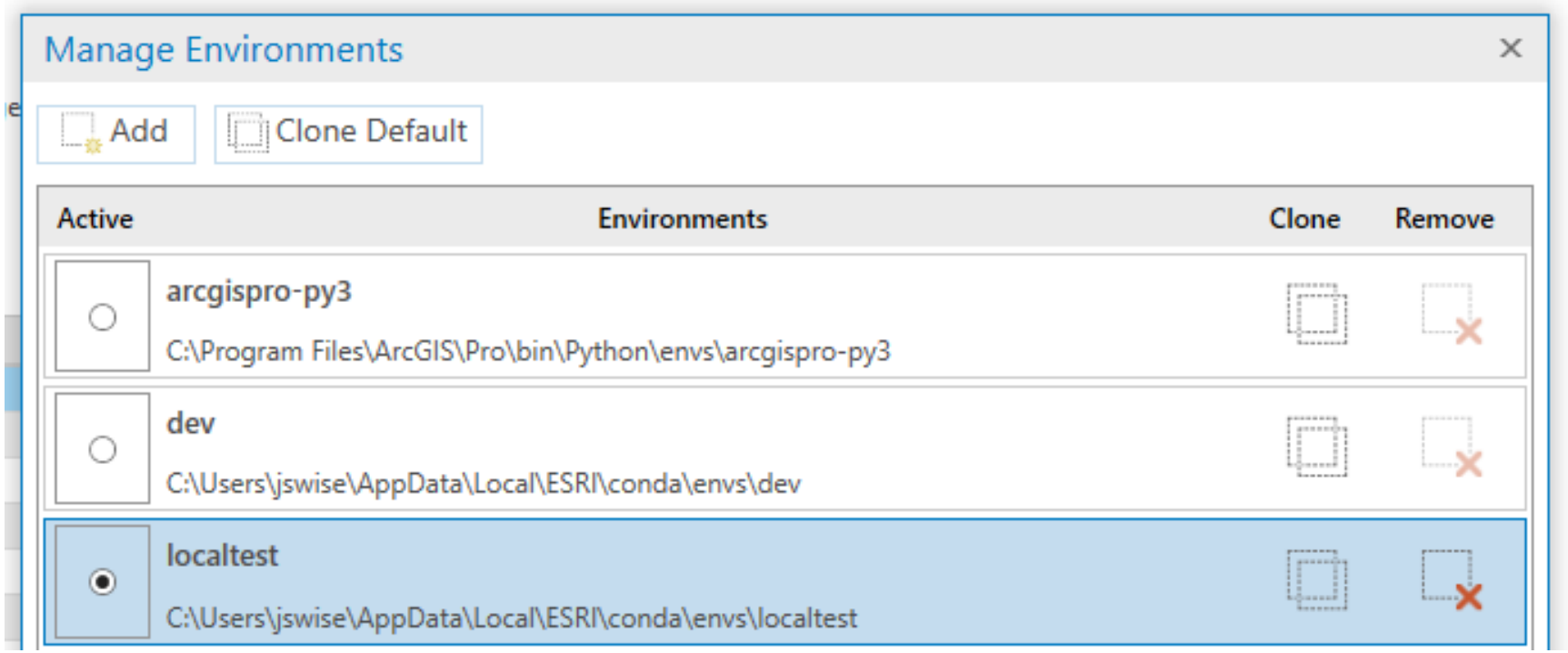
You should be able to open the toolbox in Pro.



Change environments.

Activate localtest in Pro.

Restart Pro.



Your toolbox is now broken.

The localtest environment doesn’t know about your code.

Python Errors

```
Traceback (most recent call last):
  File "<string>", line 1, in <module>
ModuleNotFoundError: No module named 'keenepyt'
```

MyTools

KeenePYT.pyt

Locators

Python

```
import keenepyt
Traceback (most recent call last):
  File "<string>", line 1, in <module>
ModuleNotFoundError: No module named 'keenepyt'
```

Look at the toolbox code.

It’s short.

It imports *keenepyt.tools.toolfactory* and runs the *get_tools* method.

```
1   from keenepyt.tools.toolfactory import * # pylint: disable=unused-wildcard-import
2
3   class Toolbox(object):
4       def __init__(self):
5           """Define the toolbox (the name of the toolbox is the name of the
6           .pyt file)."""
7           self.label = 'KeenePYT Tools'
8           self.alias = 'KeenePYT'
9           self.description = 'Demonstration code from a workshop at Spring NEARC 2019.'
10
11          self.tools = get_tools()
12
```

Look at the toolbox code.

The *get_tools* method in *keenepyt.tools.toolfactory* provides a list of tool classes.

Some are commented out so we can have fun adding them later.

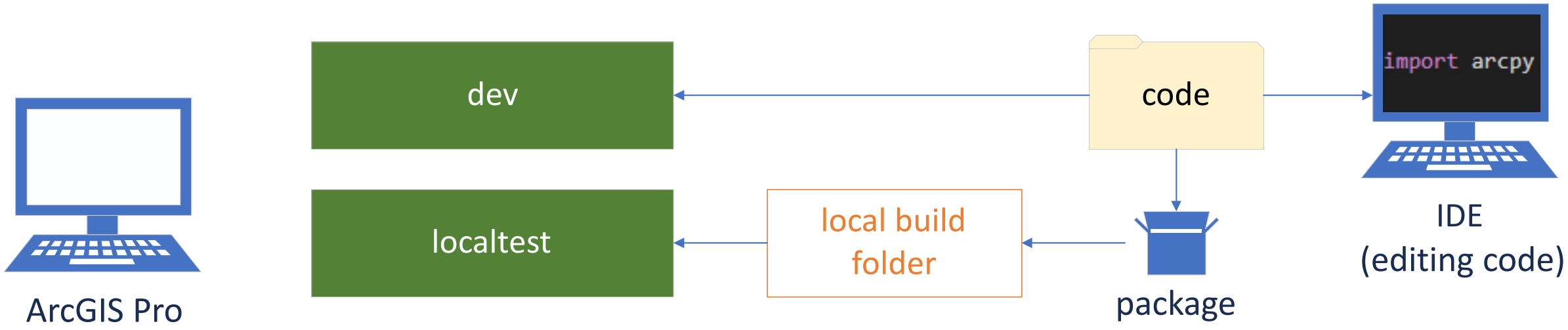
```
def get_tools():
    return [
        # GetAirepsLocal,
        # HelloInputLocal,
        HelloLocal
    ]
```

Why is it broken?

In **create-dev-envs.bat**, we told the localtest environment where to find a conda channel instead of the code. There's no package in the channel yet.

```
REM Add the build channel where the package will be.
CALL activate localtest
CALL conda config --env --add channels %build_dir%
```

Now we need to build the package.



Run build.bat.

This creates a package in

C:\Projects\NEARC\conda\conda-bld\win-64.

```
1   REM Initialize the variables, including the
2   CALL config.bat
3
4   REM Build the package.
5   CD %build_scripts%
6   CALL activate base
7   CALL conda build --py 3.6 --croot "%build_d
8   CALL conda build purge
9
10  PAUSE
```

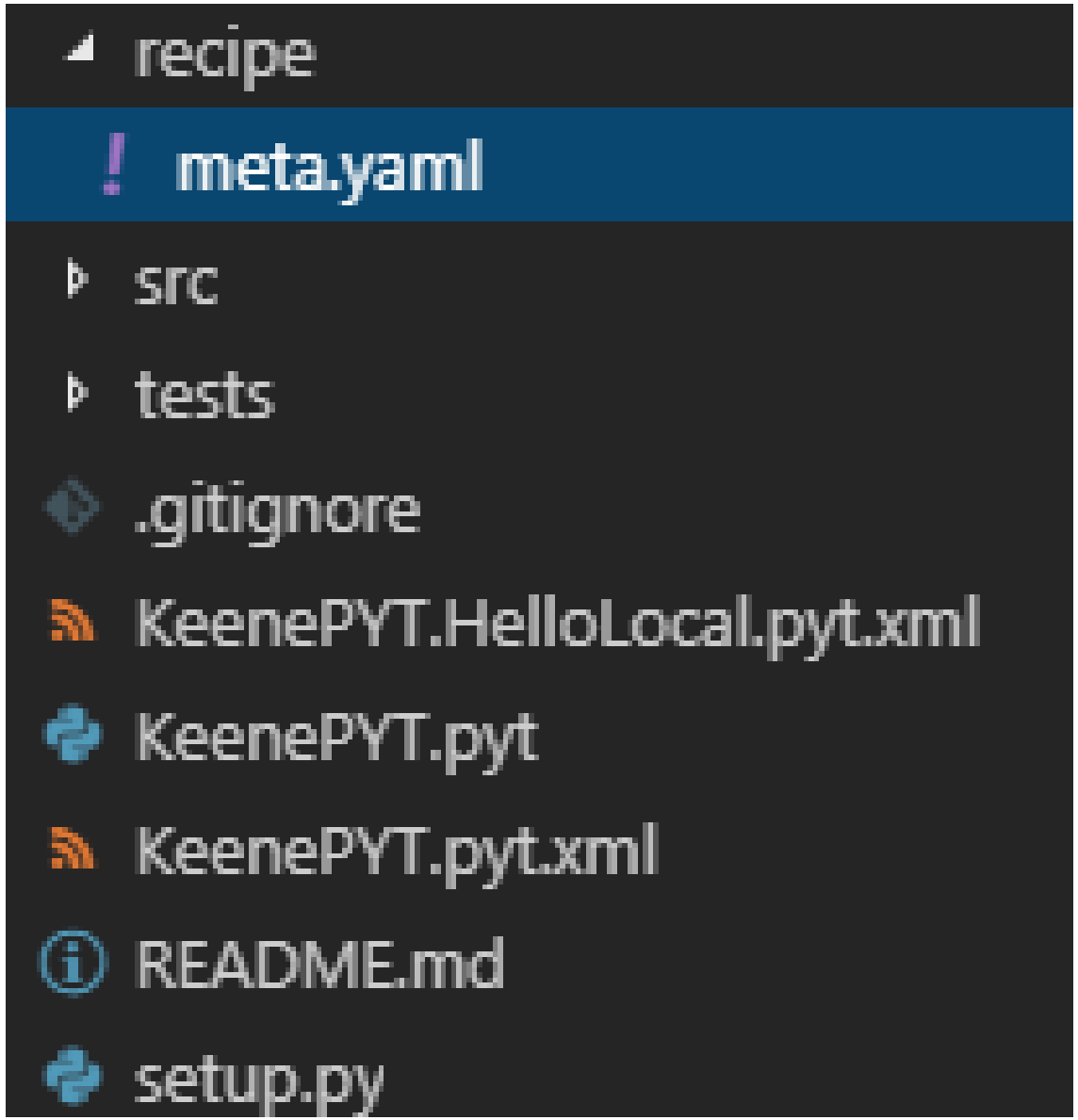
« OSDisk (C:) ▸ Projects ▸ NEARC ▸ code ▸ keenepyt ▸ batch

Name	Date modified
build.bat	5/12/2019 8:57 AM
config.bat	5/12/2019 9:47 AM
copy-from-flash.bat	5/11/2019 5:51 PM
create-dev-envs.bat	5/12/2019 10:25 AM
create-main-env.bat	5/12/2019 9:14 AM
deploy.bat	5/12/2019 9:00 AM
install-conda.bat	5/11/2019 5:51 PM
unzip.bat	5/11/2019 5:51 PM

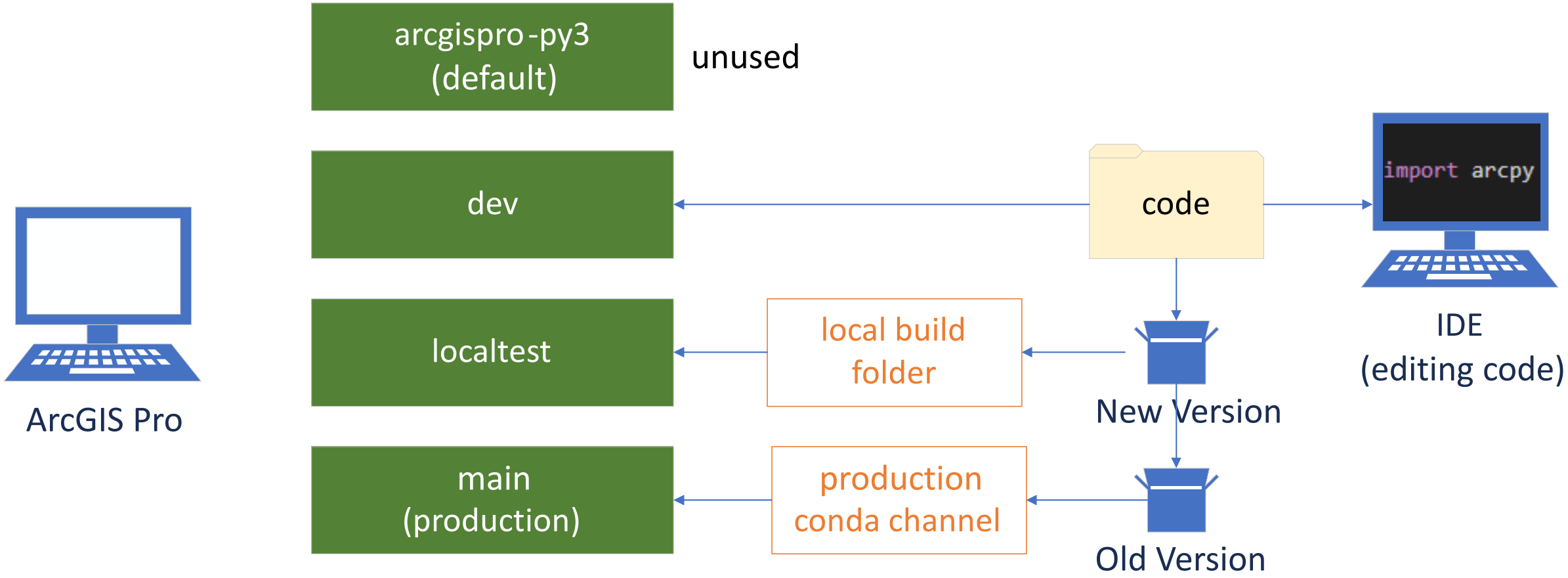
Run build.bat.

The build process depends on **recipe\meta.yaml** and **setup.py**.

We're using a separate conda installation, because we can't do it in ArcGIS Pro's conda installation.



Our environments (review)

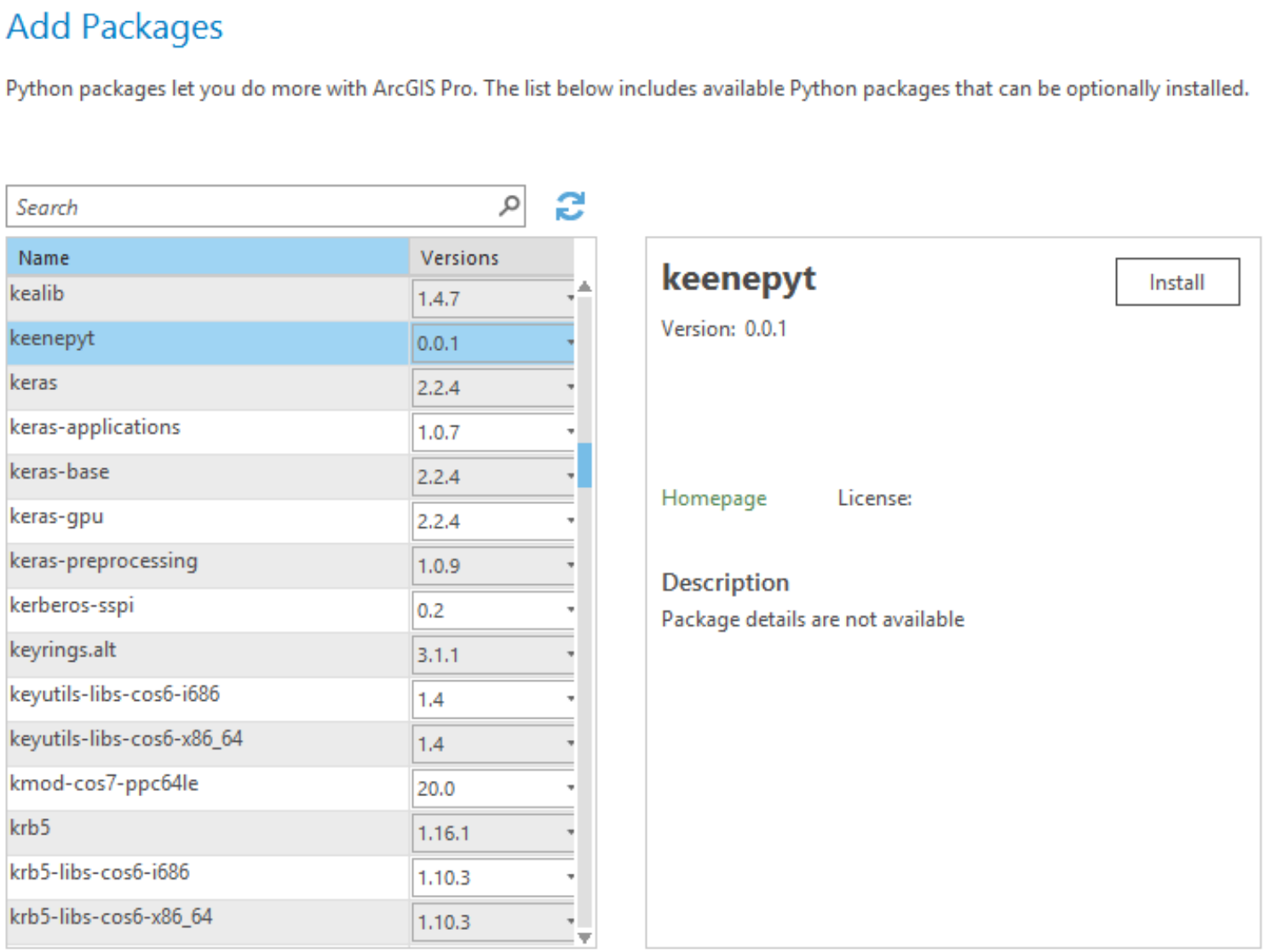


+build environment (separate conda installation)

Install the package.

You should now be able to install keenepyt in localtest.

This will fix the toolbox.

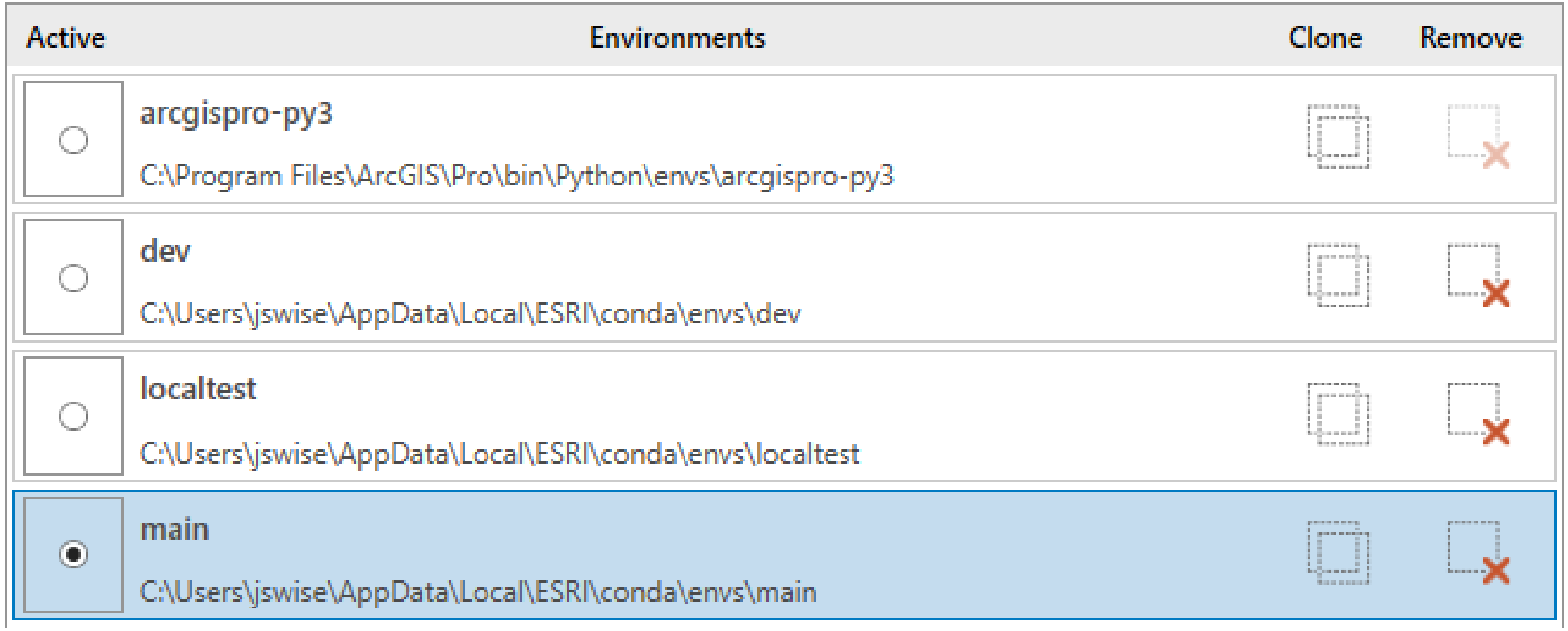


Terracon

Create the production environment.

In **C:\Projects\NEARC\code\keenepyt\batch**, run **create-main-env.bat.**

This will clone your default Python environment another new environment, *main*.



Deploy the package.

Run **deploy.bat**.

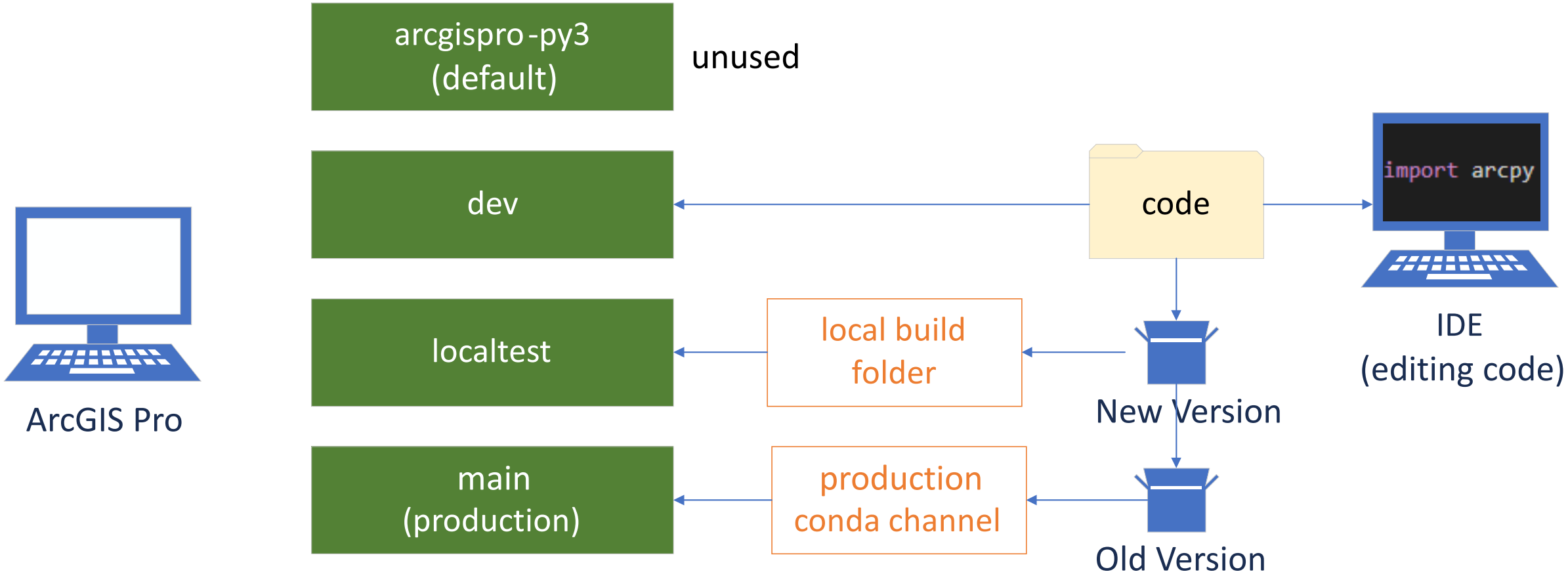
This creates a new channel, **C:\Projects\NEARC\channel**.

In real life, this would be on a server that your coworkers can see.

« OSDisk (C:) ▸ Projects ▸ NEARC ▸ code ▸ keenepyt ▸ batch

Name	Date modified
build.bat	5/12/2019 8:57 AM
config.bat	5/12/2019 9:47 AM
copy-from-flash.bat	5/11/2019 5:51 PM
create-dev-envs.bat	5/12/2019 10:25 AM
create-main-env.bat	5/12/2019 9:14 AM
deploy.bat	5/12/2019 9:00 AM
install-conda.bat	5/11/2019 5:51 PM
unzip.bat	5/11/2019 5:51 PM

Our environments (review)



+build environment (separate conda installation)

Install the package in main.

In real life, you might want to name this environment with the name of your organization.

Improve the code.

Add another tool by uncommenting it in *src.keenepyt.tools.toolfactory.get_tools*.

```
30  def get_tools():
31      return [
32          # GetAirepsLocal,
33          HelloInputLocal,
34          HelloLocal
35      ]
```

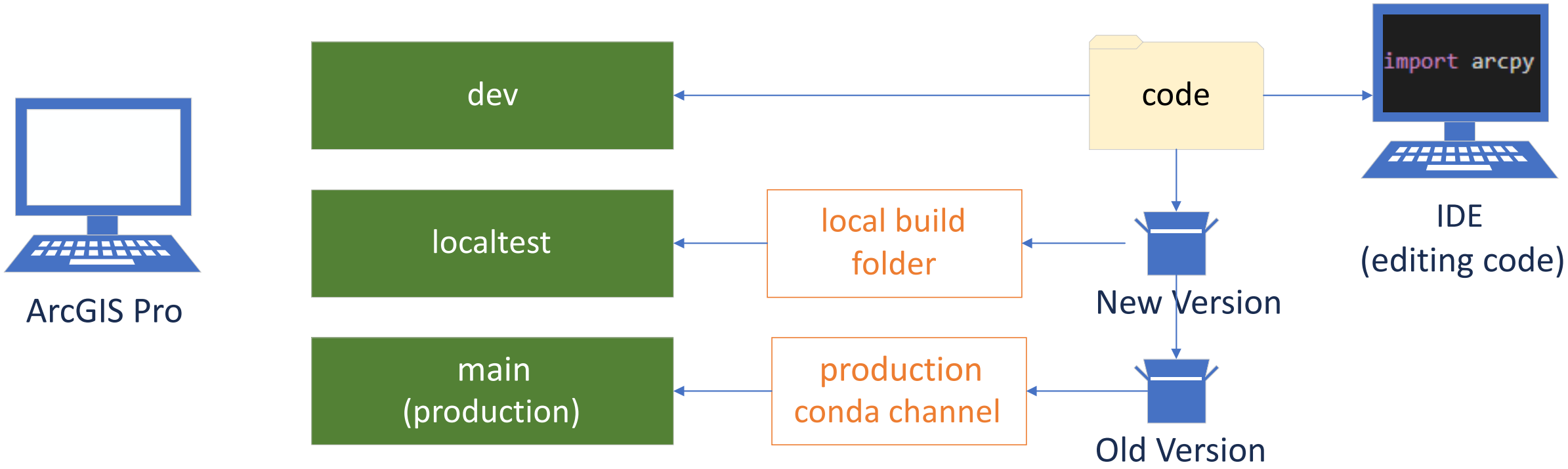
Improve the code.

In config.bat, update the version number from 0.0.1 to 0.0.2.

```
1   ECHO OFF
2
3   SET version_number=0.0.2
4   ECHO Version number: %version_number%
5
```

Run build.bat again.

Now localtest is different from main.

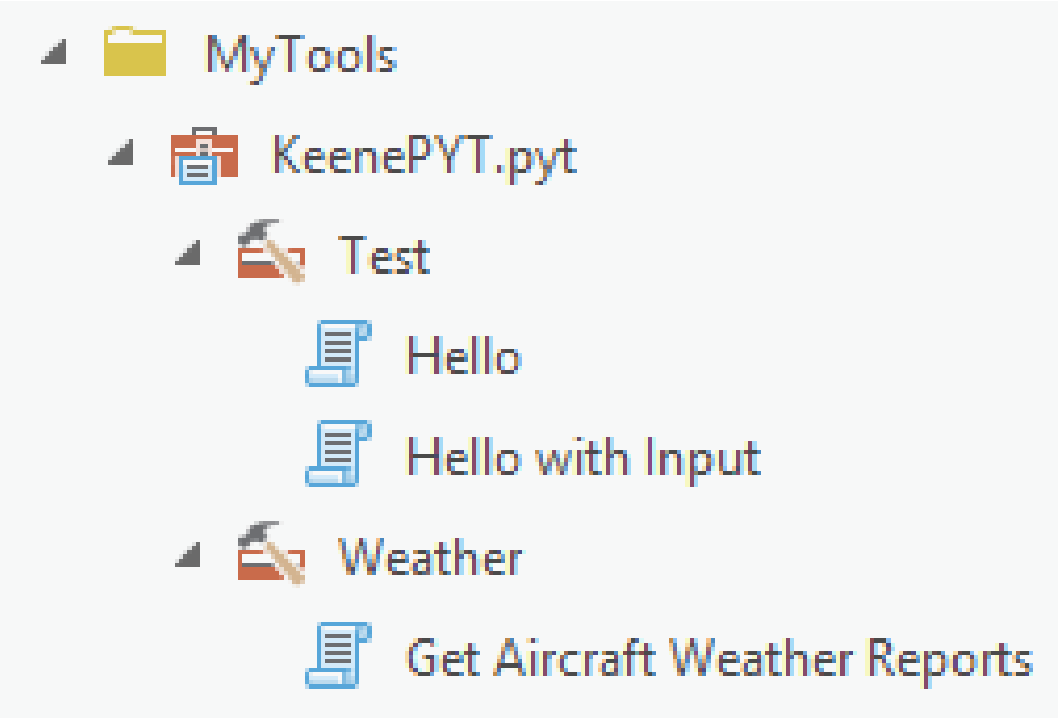


Add another tool.

Now dev, localtest, and main are all different.

Restart Pro to see the change.

```
30  def get_tools():
31      return [
32          GetAirepsLocal,
33          HelloInputLocal,
34          HelloLocal
35      ]
```



Add another tool.

Update the version number again.

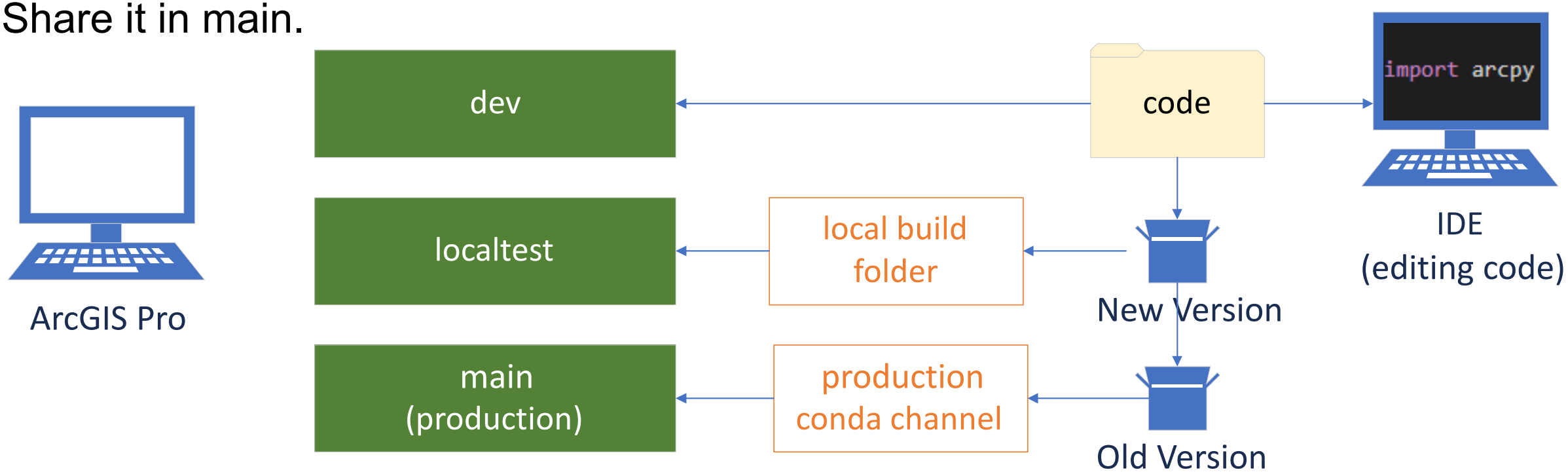
```
ECHO OFF

SET version_number=0.0.3
ECHO Version number: %version_number%

```

Development cycle

Write & test code in dev.

Test the package in localtest.

Share it in main.



More Python fun

Check out the log file.

This PC ▸ OSDisk (C:) ▸ Users ▸ jswise ▸ AppData ▸ Local ▸ KeenePYT ▸ Logs

Name	Date modified	Type
KeeneLog	5/12/2019 11:25 AM	File

```
2019-05-12 11:12:19,334 INFO:  Caller: ArcGIS
2019-05-12 11:13:36,651 INFO:  Writing C:\Users\jswise\Documents\ArcGIS\Projects\MyProject\MyProject.gdb\Aireps.
2019-05-12 11:25:20,334 INFO:  ____________________________________________________________
```

More Python fun

Use PyTest to run unit tests.

File Edit Selection View Go Debug Terminal Help — test_getaireps.py - keenepyt - Visual Studio Code

TEST: PYTHON

- tests/test_avwx.py
  - test_get_aireps
  - test_get_aireps_text
  - test_write_aireps_fc
- tests\testcore
- tests\testtools
  - tests/testtools/test_getaireps.py
    - test_execute
  - tests/testtools/test_hello.py
  - tests/testtools/test_helloinput.py

config.bat | copy-from-flash.bat | create-dev-envs.bat | build.bat | setup.py | meta.ya

```python
import logging
import pytest

from keenepyt.tools.getaireps import GetAireps

GDB = r'c:\Projects\NEARC\Maps\KeenePYT\Default.gdb'

@pytest.fixture
def victim():
    return GetAireps(log_level=logging.INFO)

X Run Test | X Debug Test
def test_execute(victim):
```

test_getaireps.py 1 of 1 problem

```
Fail: RuntimeError: The geodatabase does not exist (c:\Projects\NEARC\Maps\KeenePYT\Default.gdb).

test_getaireps.py(12, 5): test_execute
test_getaireps.py(15, 5): victim.execute(params, None)
getaireps.py(41, 9): fc = aviation_weather.write_aireps_fc(gdb, fc_name)
avwx.py(75, 13): raise RuntimeError('The geodatabase does not exist ({}).'.format(gdb))
```

```python
    params = victim.getParameterInfo()
```

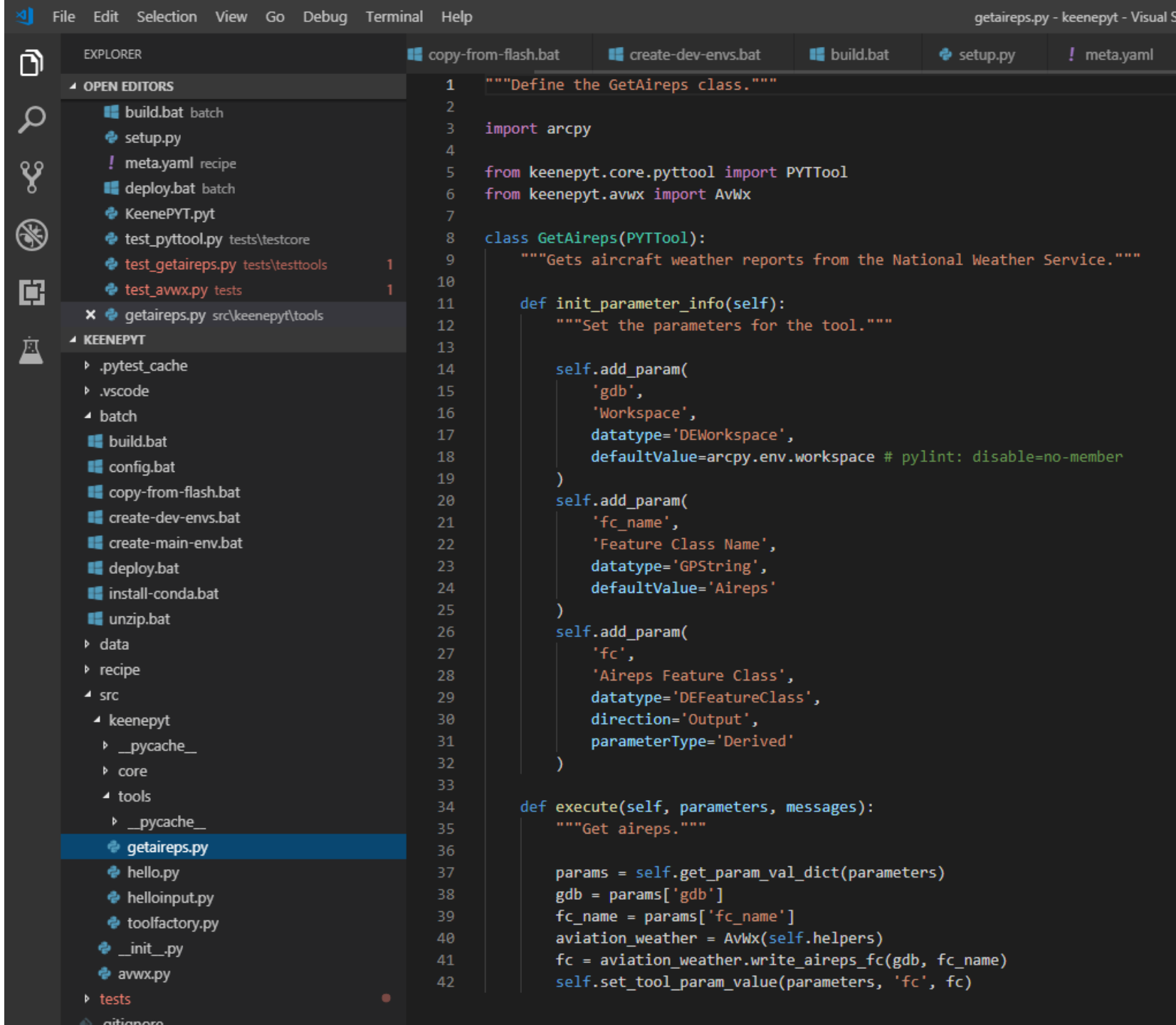
Terracon

More Python fun

See how these tools work.

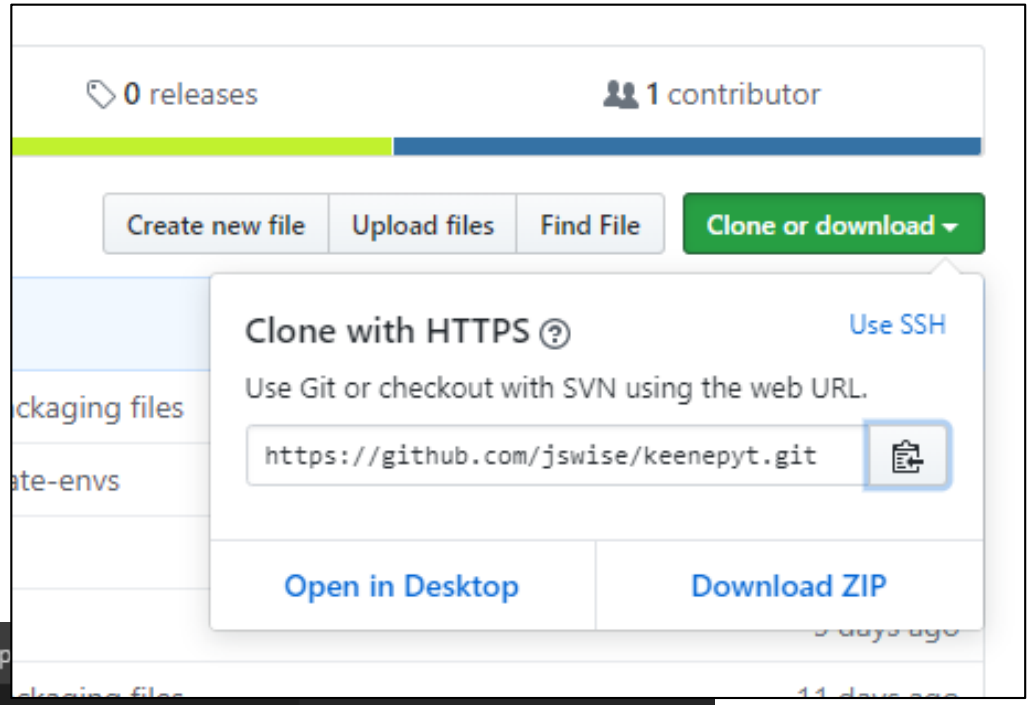
They're not like the examples you'll see elsewhere.

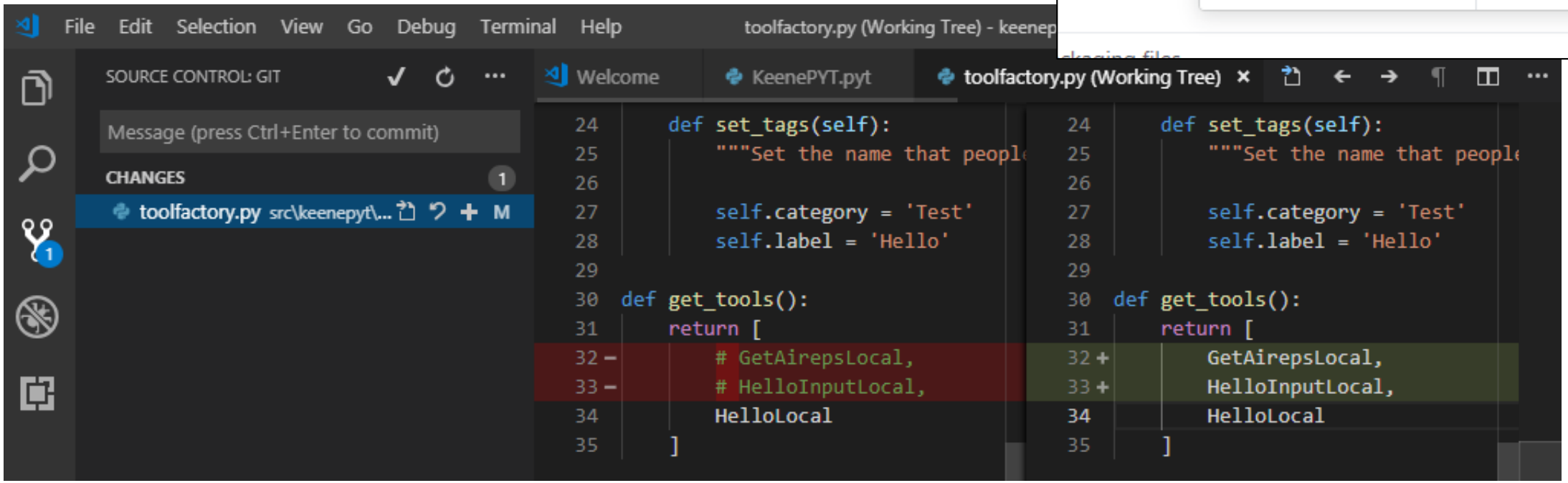
Terracon



More Python fun

Use a Git repository.





Jason Wise

jason.wise@Terracon.com
https://github.com/jswise

Terracon

RESPONSIVE.
RESOURCEFUL.
RELIABLE.