Creating and Distributing Python Packages

Starting at 12:05pm

Tyler Collins
@andesha

SHARCNET
Brock University

Wed, September 21st | 12pm EST

Today’s Structure

- Slides will be available online after the session
    - No need to take strict notes on commands used
- Recording will also be posted on the SHARCNET YouTube account
    - You can pause and follow along later if you’d like
- Open questions will be allowed at the end of the session
    - Chat questions will be monitored during

Don’t memorize every little command given - follow the principle of what’s happening instead!

Today’s Outline

1. Terminology
2. Reminder of how to install a Python package
3. What do Python packages look like?
4. What makes a good package?
5. Intro to Cookiecutter and usage
6. Interactions with GitHub
7. Releases and building wheels
8. Deploying
9. Takeaways and questions

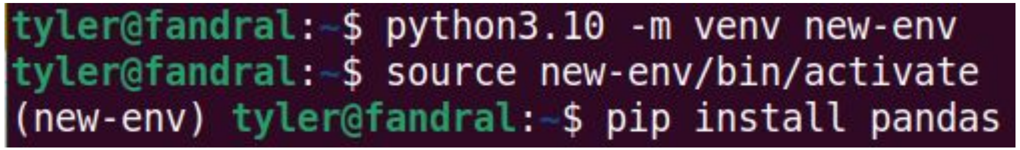
Terminology

So everyone can get on the same page:

- **Package:** Some directory (folder) of grouped code
- **Distribution Package:** Sharable form of a regular package, typically a wheel
- **Wheel:** Precompiled and ready-to-use distribution package
- **Pip:** Command line tool for installing python packages
- **Virtual Environment:** Isolated installation of packages and Python version
- **PyPI:** Python Package Index, typically where “pip” reads from by default

How do we install packages in Python?

Installing “pandas” in a fresh virtual environment:

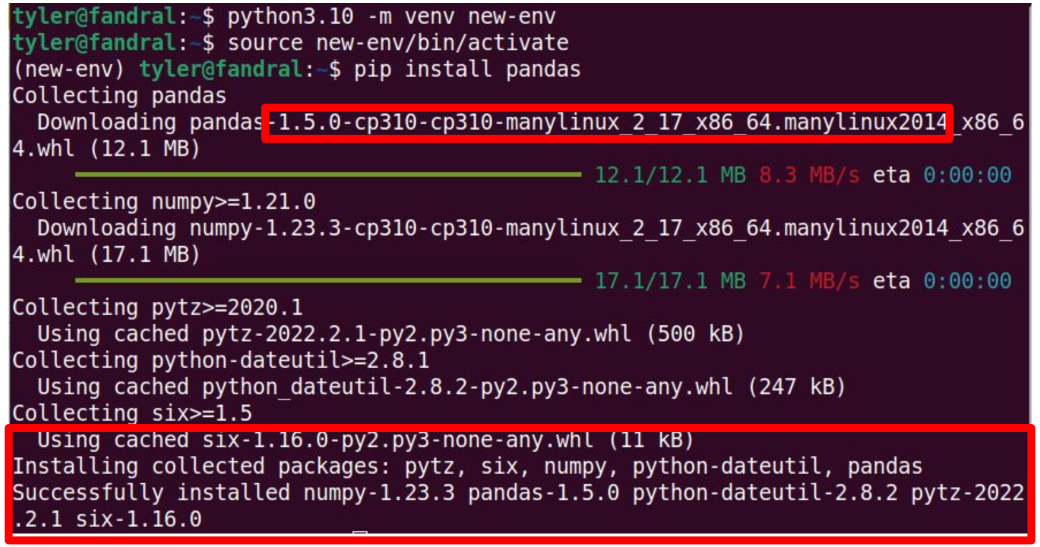


```
tyler@fandral:~$ python3.10 -m venv new-env
tyler@fandral:~$ source new-env/bin/activate
(new-env) tyler@fandral:~$ pip install pandas
```

Easy for us now, but how did it get to that point?

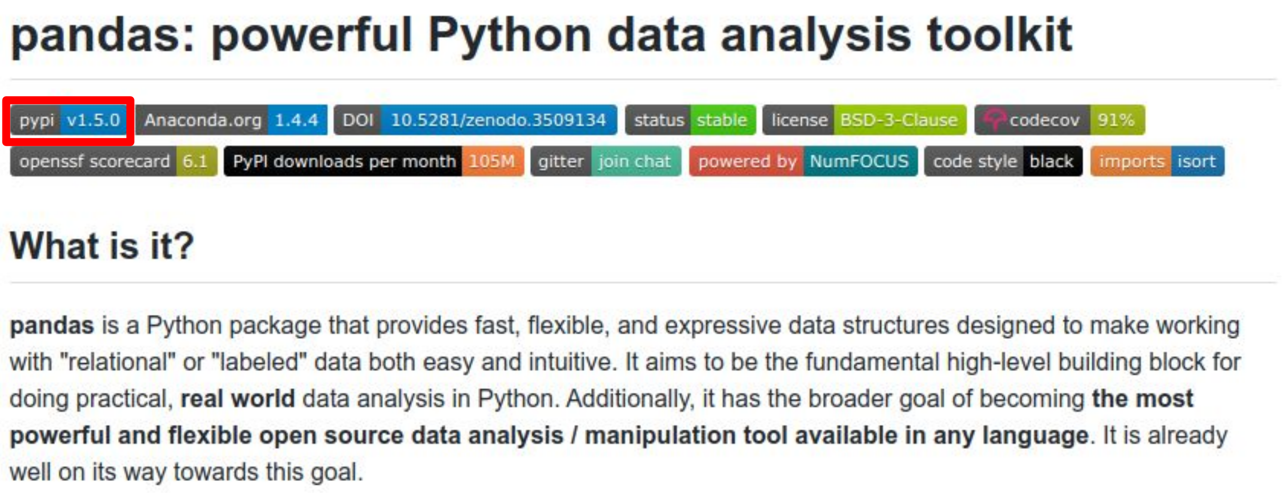
How do we install packages in Python?

And what’s all of this anyway?



```
tyler@fandral:~$ python3.10 -m venv new-env
tyler@fandral:~$ source new-env/bin/activate
(new-env) tyler@fandral:~$ pip install pandas
Collecting pandas
  Downloading pandas-1.5.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_6
4.whl (12.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.1/12.1 MB 8.3 MB/s eta 0:00:00
Collecting numpy>=1.21.0
  Downloading numpy-1.23.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_6
4.whl (17.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 17.1/17.1 MB 7.1 MB/s eta 0:00:00
Collecting pytz>=2020.1
  Using cached pytz-2022.2.1-py2.py3-none-any.whl (500 kB)
Collecting python-dateutil>=2.8.1
  Using cached python_dateutil-2.8.2-py2.py3-none-any.whl (247 kB)
Collecting six>=1.5
  Using cached six-1.16.0-py2.py3-none-any.whl (11 kB)
Installing collected packages: pytz, six, numpy, python-dateutil, pandas
Successfully installed numpy-1.23.3 pandas-1.5.0 python-dateutil-2.8.2 pytz-2022
.2.1 six-1.16.0
```

What do modern packages look like?



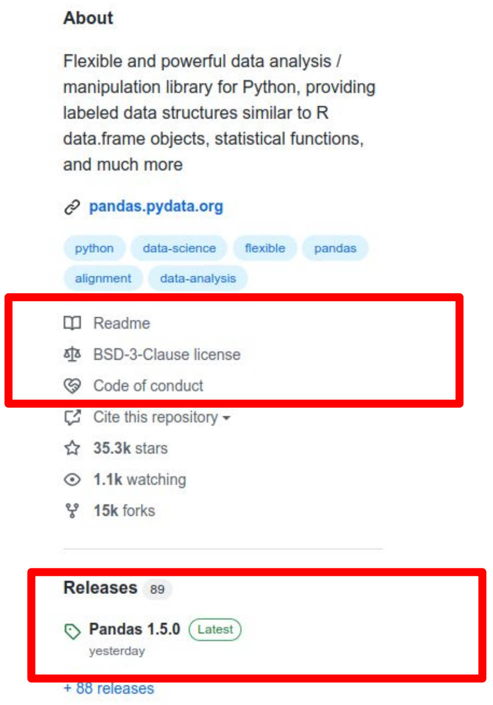
pandas: powerful Python data analysis toolkit

pypi v1.5.0 | Anaconda.org 1.4.4 | DOI 10.5281/zenodo.3509134 | status stable | license BSD-3-Clause | codecov 91%

openssf scorecard 6.1 | PyPI downloads per month 105M | gitter join chat | powered by NumFOCUS | code style black | imports isort

What is it?

pandas is a Python package that provides fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, **real world** data analysis in Python. Additionally, it has the broader goal of becoming **the most powerful and flexible open source data analysis / manipulation tool available in any language**. It is already well on its way towards this goal.



About

Flexible and powerful data analysis / manipulation library for Python, providing labeled data structures similar to R data.frame objects, statistical functions, and much more

pandas.pydata.org

python · data-science · flexible · pandas · alignment · data-analysis

Readme
BSD-3-Clause license
Code of conduct
Cite this repository
35.3k stars
1.1k watching
15k forks

Releases 89

Pandas 1.5.0 Latest
yesterday

+ 88 releases

What makes a good package then?

Wants versus needs is dependent on use case, however for today:

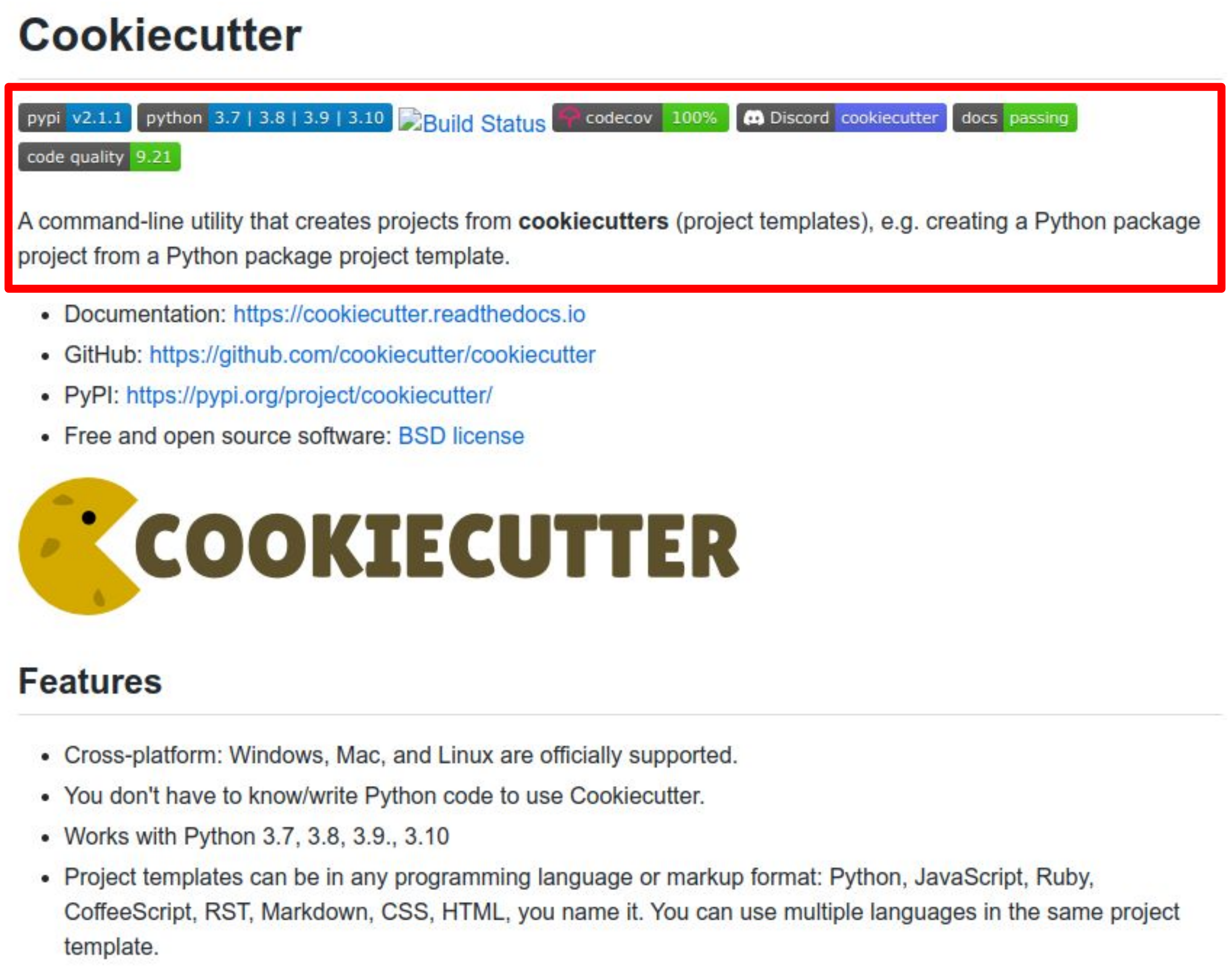
- Needs:
  - Version control and hosting
  - Documentation
  - Shareability and releases
  - Contributing/licensing
- Wants (not strictly covered today):
  - Testing
  - CI/CD
  - Issue management, etc

/r/DiWHY

We're in Python, we should **always** avoid reinventing the wheel

Does something exist that can manage this for us? Of course!

Today's talk focuses on Cookiecutter and its interactions with GitHub



Cookiecutter

pypi v2.1.1 python 3.7 | 3.8 | 3.9 | 3.10 Build Status codecov 100% Discord cookiecutter docs passing
code quality 9.21

A command-line utility that creates projects from **cookiecutters** (project templates), e.g. creating a Python package project from a Python package project template.

- Documentation: https://cookiecutter.readthedocs.io
- GitHub: https://github.com/cookiecutter/cookiecutter
- PyPI: https://pypi.org/project/cookiecutter/
- Free and open source software: BSD license

COOKIECUTTER

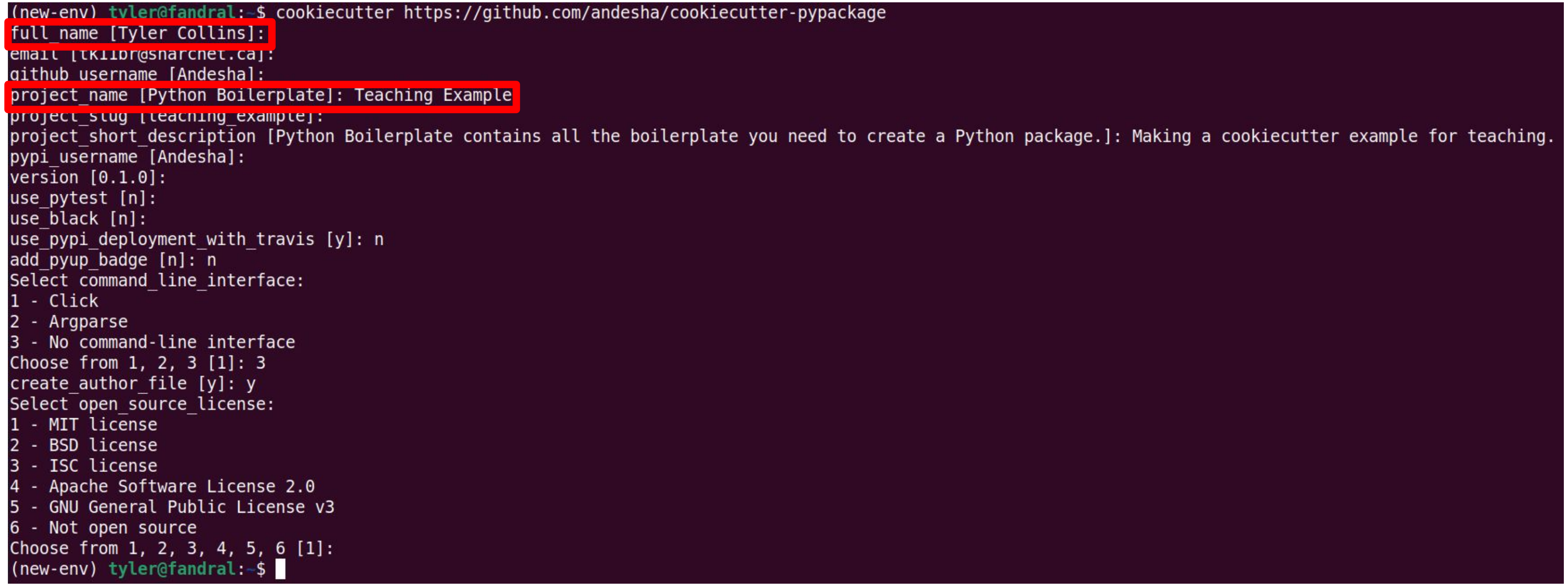
Features

- Cross-platform: Windows, Mac, and Linux are officially supported.
- You don't have to know/write Python code to use Cookiecutter.
- Works with Python 3.7, 3.8, 3.9., 3.10
- Project templates can be in any programming language or markup format: Python, JavaScript, Ruby, CoffeeScript, RST, Markdown, CSS, HTML, you name it. You can use multiple languages in the same project template.

Interacting with Cookiecutter

1. Install Cookiecutter via pip in a global or local environment
    a. “pip install cookiecutter”
2. Point Cookiecutter at a reference template
    a. “cookiecutter https://github.com/andesha/cookiecutter-pypackage”
3. Answer the prompts as it relates to your project
    a. **Pointing it at your own fork lets you change the defaults!**

Interacting with Cookiecutter

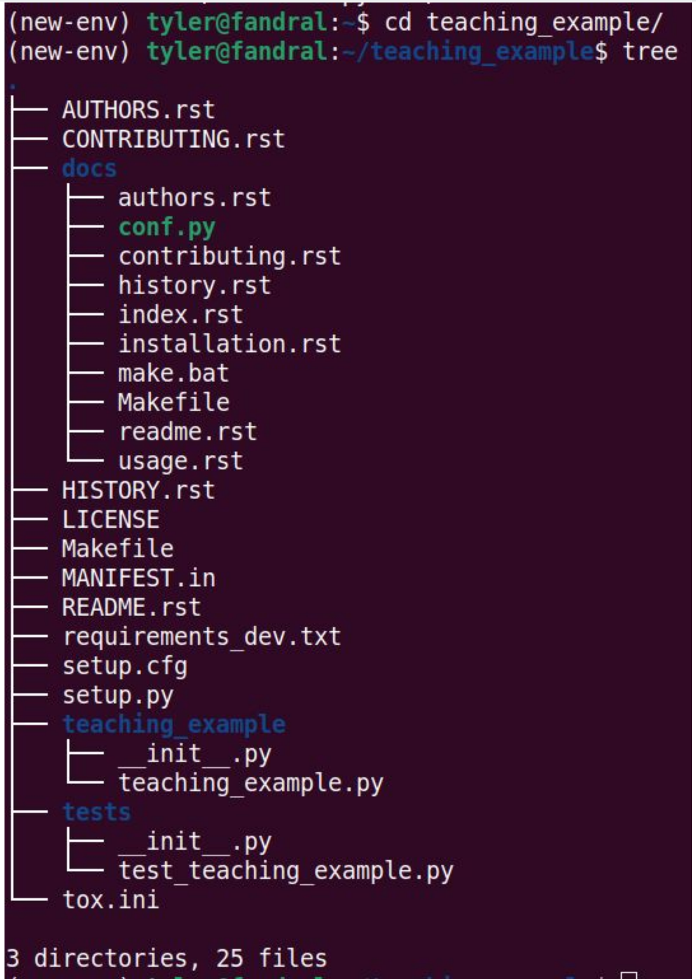


```
(new-env) tyler@fandral:~$ cookiecutter https://github.com/andesha/cookiecutter-pypackage
full_name [Tyler Collins]:
email [tkilbr@sharcnet.ca]:
github_username [Andesha]:
project_name [Python Boilerplate]: Teaching Example
project_slug [teaching_example]:
project_short_description [Python Boilerplate contains all the boilerplate you need to create a Python package.]: Making a cookiecutter example for teaching.
pypi_username [Andesha]:
version [0.1.0]:
use_pytest [n]:
use_black [n]:
use_pypi_deployment_with_travis [y]: n
add_pyup_badge [n]: n
Select command_line_interface:
1 - Click
2 - Argparse
3 - No command-line interface
Choose from 1, 2, 3 [1]: 3
create_author_file [y]: y
Select open_source_license:
1 - MIT license
2 - BSD license
3 - ISC license
4 - Apache Software License 2.0
5 - GNU General Public License v3
6 - Not open source
Choose from 1, 2, 3, 4, 5, 6 [1]:
(new-env) tyler@fandral:~$
```

What do we get from Cookiecutter?

- Docs folders for uploading to services
- History, license, README, authorship, and contributing references
- Requirements file is typically for what developers need to work with your package
- Manifest is for shipping things with your package that are not code
- Tests and tox are for things like CI/CD
- “teaching_example” is the code folder

What’s most important is the “**setup.py**” file



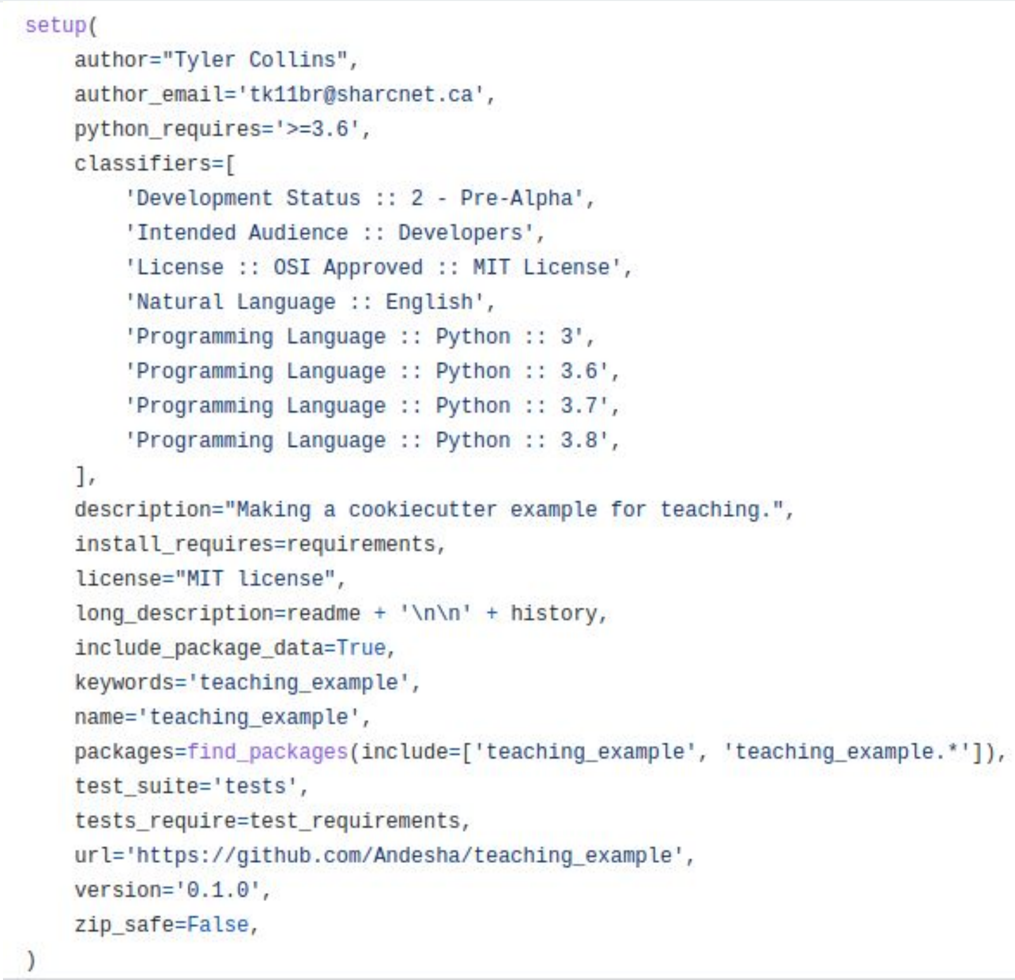
```
(new-env) tyler@fandral:~$ cd teaching_example/
(new-env) tyler@fandral:~/teaching_example$ tree
.
├── AUTHORS.rst
├── CONTRIBUTING.rst
├── docs
│   ├── authors.rst
│   ├── conf.py
│   ├── contributing.rst
│   ├── history.rst
│   ├── index.rst
│   ├── installation.rst
│   ├── make.bat
│   ├── Makefile
│   ├── readme.rst
│   └── usage.rst
├── HISTORY.rst
├── LICENSE
├── Makefile
├── MANIFEST.in
├── README.rst
├── requirements_dev.txt
├── setup.cfg
├── setup.py
├── teaching_example
│   ├── __init__.py
│   └── teaching_example.py
├── tests
│   ├── __init__.py
│   └── test_teaching_example.py
└── tox.ini

3 directories, 25 files
```

What's in a setup.py?

The image on the right is a code snippet

Specifies how to build your package and various metadata properties about it

Also allows you to specify dependencies



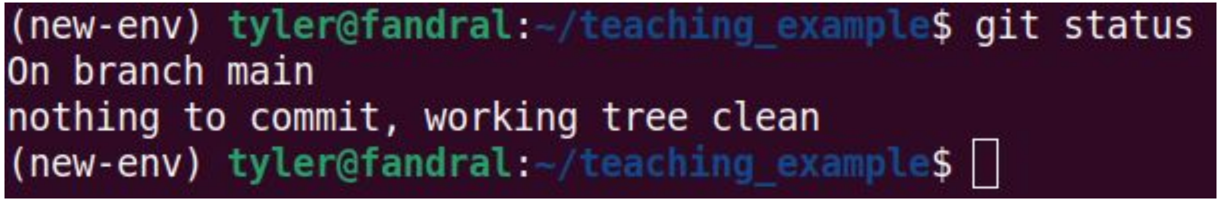
```
setup(
    author="Tyler Collins",
    author_email='tk11br@sharcnet.ca',
    python_requires='>=3.6',
    classifiers=[
        'Development Status :: 2 - Pre-Alpha',
        'Intended Audience :: Developers',
        'License :: OSI Approved :: MIT License',
        'Natural Language :: English',
        'Programming Language :: Python :: 3',
        'Programming Language :: Python :: 3.6',
        'Programming Language :: Python :: 3.7',
        'Programming Language :: Python :: 3.8',
    ],
    description="Making a cookiecutter example for teaching.",
    install_requires=requirements,
    license="MIT license",
    long_description=readme + '\n\n' + history,
    include_package_data=True,
    keywords='teaching_example',
    name='teaching_example',
    packages=find_packages(include=['teaching_example', 'teaching_example.*']),
    test_suite='tests',
    tests_require=test_requirements,
    url='https://github.com/Andesha/teaching_example',
    version='0.1.0',
    zip_safe=False,
)
```

How do we link this up to GitHub?

First, let’s make the new project structure a Git repository:

1. Enter the directory with “cd teaching_example”
2. Run “git init”
    a. You may be given a prompt about choosing your main branch name. I suggest using “main”.
3. Add all files via “git add -A”
4. Complete the first commit as you like, or with “git commit -m ‘initial commit’”

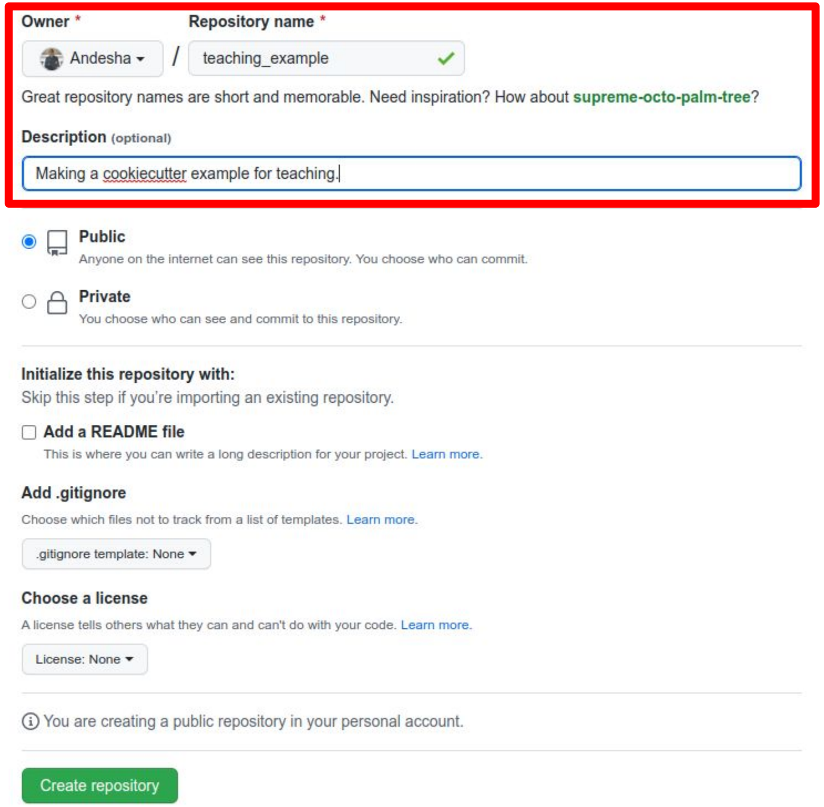
Now onto the GitHub side!



```
(new-env) tyler@fandral:~/teaching_example$ git status
On branch main
nothing to commit, working tree clean
(new-env) tyler@fandral:~/teaching_example$ 
```

Linking up with GitHub

1. Fill in the repository name with what you used during the Cookiecutter prompts
2. Do the same for the description
3. Leave the rest alone - Cookiecutter did it for you!
4. Create the repository



Owner * Andesha / Repository name * teaching_example

Great repository names are short and memorable. Need inspiration? How about **supreme-octo-palm-tree**?

Description (optional)

Making a cookiecutter example for teaching.

● **Public** Anyone on the internet can see this repository. You choose who can commit.

○ **Private** You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

☐ **Add a README file**
This is where you can write a long description for your project. Learn more.

Add .gitignore
Choose which files not to track from a list of templates. Learn more.

.gitignore template: None

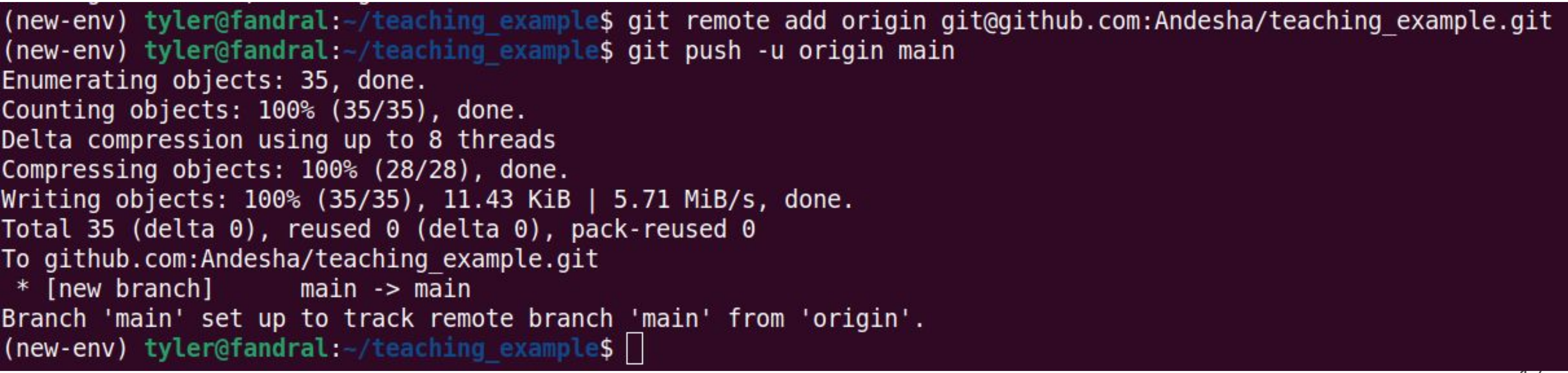
Choose a license
A license tells others what they can and can't do with your code. Learn more.

License: None

You are creating a public repository in your personal account.

Last step with GitHub...

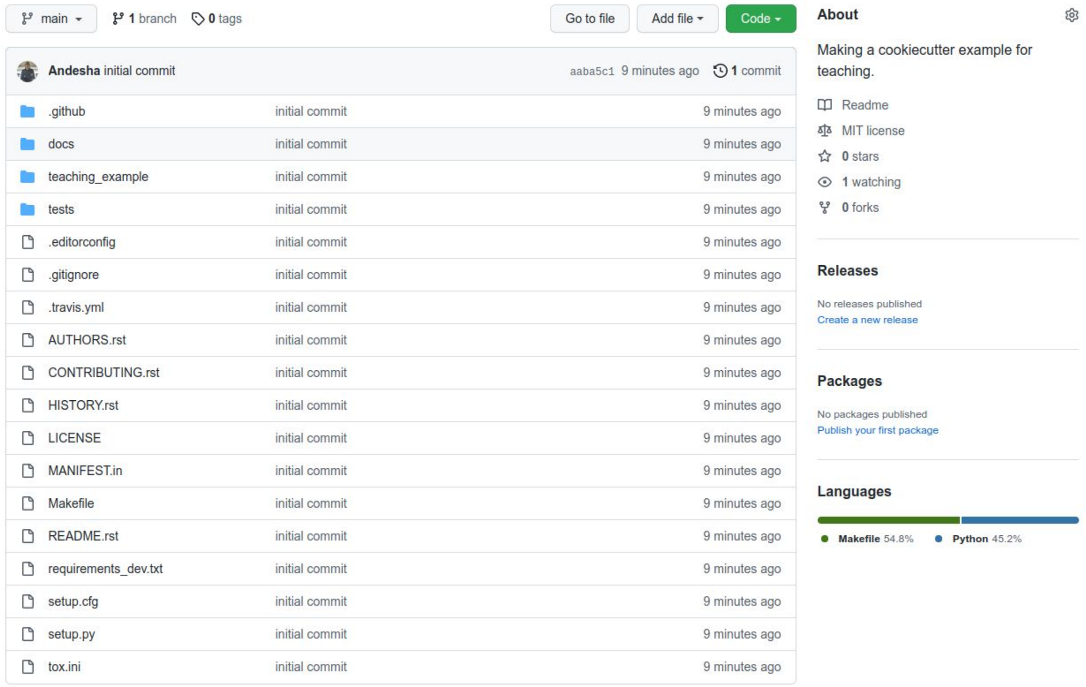
Follow the second prompt adding a new remote, and pushing to it:



```
(new-env) tyler@fandral:~/teaching_example$ git remote add origin git@github.com:Andesha/teaching_example.git
(new-env) tyler@fandral:~/teaching_example$ git push -u origin main
Enumerating objects: 35, done.
Counting objects: 100% (35/35), done.
Delta compression using up to 8 threads
Compressing objects: 100% (28/28), done.
Writing objects: 100% (35/35), 11.43 KiB | 5.71 MiB/s, done.
Total 35 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:Andesha/teaching_example.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
(new-env) tyler@fandral:~/teaching_example$ 
```

How does it look?

Check for yourself at my own profile, here later



main | 1 branch | 0 tags | Go to file | Add file | Code

Andesha initial commit aaba5c1 9 minutes ago 1 commit

.github	initial commit	9 minutes ago
docs	initial commit	9 minutes ago
teaching_example	initial commit	9 minutes ago
tests	initial commit	9 minutes ago
.editorconfig	initial commit	9 minutes ago
.gitignore	initial commit	9 minutes ago
.travis.yml	initial commit	9 minutes ago
AUTHORS.rst	initial commit	9 minutes ago
CONTRIBUTING.rst	initial commit	9 minutes ago
HISTORY.rst	initial commit	9 minutes ago
LICENSE	initial commit	9 minutes ago
MANIFEST.in	initial commit	9 minutes ago
Makefile	initial commit	9 minutes ago
README.rst	initial commit	9 minutes ago
requirements_dev.txt	initial commit	9 minutes ago
setup.cfg	initial commit	9 minutes ago
setup.py	initial commit	9 minutes ago
tox.ini	initial commit	9 minutes ago

About

Making a cookiecutter example for teaching.

Readme
MIT license
0 stars
1 watching
0 forks

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Languages

● Makefile 54.8% ● Python 45.2%

Status Check-in

What do we have:

- GitHub integration and source hosting
- Licensing, READMEs, etc
- Modern project layout automation
  - Docs will be automatically generated with some settings

What do we have left:

- Releases and sharing
- PyPI and installation via pip

What's left is easy, I promise!

Releases

There are tools that can automate this procedure, but doing it manually is more fun for me:

1. Complete changes to your package such that it’s in some new versioned “state”
2. Update both the HISTORY.rst and “version” variable inside of the “__init__.py”
   a. These **must** match for automation to play nice!
3. Run the following git commands to signal to GitHub that this is a stable version or release:
   a. “git tag 0.2.0” or whatever version you are moving to
   b. “git push origin --tags”
   c. GitHub will now see this is a tagged state on your repository for potential release

Onto the final step, distribution!

Building Wheels

First we need to actually make the distribution packages that will be shared

1. Ensure wheel is installed via “pip install wheel”
2. Run the following:
    a. “./setup.py sdist”
    b. “./setup.py bdist_wheel”
3. Done and done!
    a. You could share these to a friend and have them install the wheel via pip if you wanted

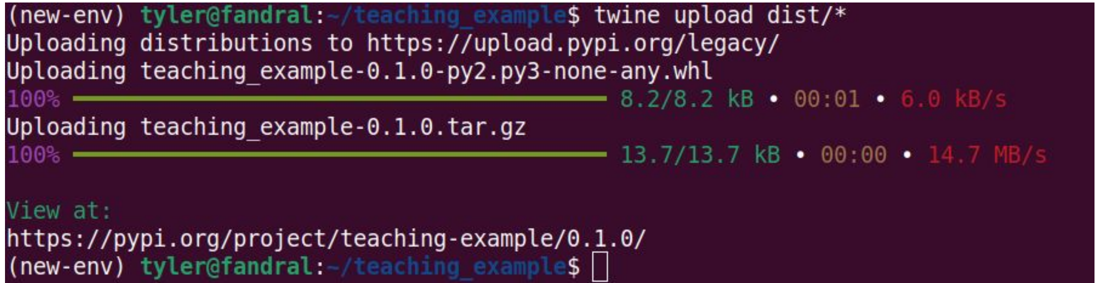
Remember - “setup.py” is our **makefile** and lets us define numerous things outlined here

Deploying to PyPI

Time to share to the world

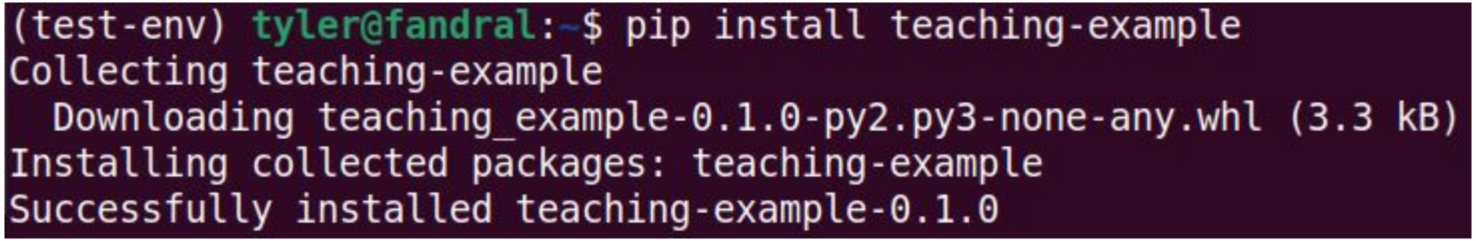
1. Make an account on PyPI
2. Set up local authentication through their prompts
   a. I like to use the settings that outline creating a ".pypirc" file
3. Ensure "twine" is installed via "pip install twine" and run the following:
   a. "twine upload dist/*"
   b. You may need to empty your old dist folder before a successful upload

Done for real this time!

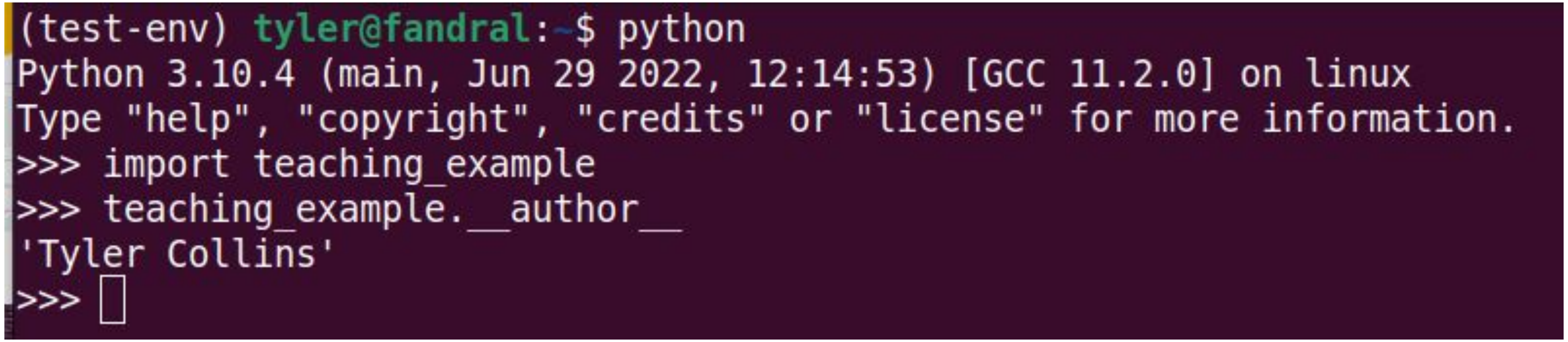


```
(new-env) tyler@fandral:~/teaching_example$ twine upload dist/*
Uploading distributions to https://upload.pypi.org/legacy/
Uploading teaching_example-0.1.0-py2.py3-none-any.whl
100% ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.2/8.2 kB • 00:01 • 6.0 kB/s
Uploading teaching_example-0.1.0.tar.gz
100% ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.7/13.7 kB • 00:00 • 14.7 MB/s

View at:
https://pypi.org/project/teaching-example/0.1.0/
(new-env) tyler@fandral:~/teaching_example$
```

Deploying to PyPI: checking if it worked!



```
(test-env) tyler@fandral:~$ pip install teaching-example
Collecting teaching-example
  Downloading teaching_example-0.1.0-py2.py3-none-any.whl (3.3 kB)
Installing collected packages: teaching-example
Successfully installed teaching-example-0.1.0
```



```
(test-env) tyler@fandral:~$ python
Python 3.10.4 (main, Jun 29 2022, 12:14:53) [GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import teaching_example
>>> teaching_example.__author__
'Tyler Collins'
>>> 
```

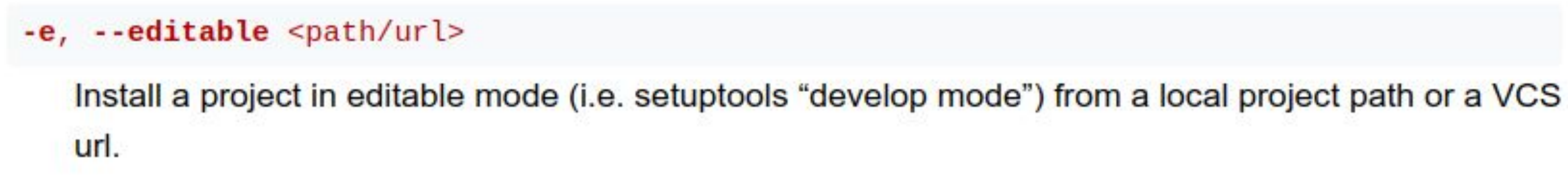
How do I develop on this structure locally?

Choosing to work within this structure **does not** mean you can no longer develop locally

Pip conveniently includes a mechanism for this via the “-e” flag when installing

For example, enter the “teaching_examples” directory followed by “pip install -e .” to locally install the package

Edits to the package are then made available to the installed environment



```
-e, --editable <path/url>
```

Install a project in editable mode (i.e. setuptools “develop mode”) from a local project path or a VCS url.

Takeaways

- Don’t reinvent the wheel when sharing code
- Use Cookiecutter to automate project boilerplate construction
- Change the defaults - it will make your life easier!
- Further reading for Travis testing and Read the Docs will be helpful

Thanks for your attention!

Questions?

Tyler Collins
tk11br@sharcnet.ca
@andesha