1. Installing Packages

What are packages and why are they needed?

Conda packages are files containing a bundle of resources: usually libraries and executables, but not always. In principle, Conda packages can include data, images, notebooks, or other assets. The command-line tool conda is used to install, remove and examine packages; other tools such as the GUI *Anaconda Navigator* also expose the same capabilities. This course focuses on the conda tool itself (you'll see use cases other than package management in later chapters).

Conda packages are most widely used with Python, but that's not all. Nothing about the Conda package format or the conda tool itself assumes any specific programming language. Conda packages can also be used for bundling libraries in other languages (like R, Scala, Julia, etc.) or simply for distributing pure binary executables generated from *any*programming language.

One of the powerful aspects of conda—both the tool and the package format—is that dependencies are taken care of. That is, when you install any Conda package, any other packages needed get installed automatically. Tracking and determining software dependencies is a hard problem that package managers like Conda are designed to solve.

A Conda package, then, is a file containing all files needed to make a given program execute correctly on a given system. Moreover, a Conda package can contain binary artifacts specific to a particular platform or operating system. Most packages (and their dependencies) are available for Windows (win-32 or win-64), for OSX (osx-64), and for Linux (linux-32 or linux-64). A small number of Conda packages are available for more specialized platforms (e.g., Raspberry Pi 2 or POWER8 LE). As a user, you do not need to specify the platform since Conda will simply choose the Conda package appropriate for the platform you are using.

Conda packages' features

- The Conda package format is programming-language and asset-type independent.
- Packages contain a description of all dependencies, all of which are installed together.
- The tool conda can be used to install, examine, or remove packages from a working system.
- Other GUI or web-based tools can be used as a wrapper for the tool conda for package management.

determine version of conda

```
1  (base) $ conda --version
```

```
2    conda 4.7.5
```

Install a conda package (I)

```
1    (base) $ conda install --help | grep package_spec
2                        [package_spec [package_spec ...]]
3      package_spec          Packages to install or update in the conda environment.
```

Install a conda package (II)

```
1    (base) $ conda install cytoolz
```

semantic versioning

Most Conda packages use a system called *semantic versioning* to identify distinct versions of a software package unambiguously.

Under semantic versioning, software is labeled with a three-part version identifier of the form MAJOR.MINOR.PATCH; the label components are non-negative integers separated by periods. Assuming all software starts at version 0.0.0

package version: conda list

```
1    (base) $ conda list
2    # packages in environment at /home/repl/miniconda:
3    ## Name                    Version                   Build  Channel
4    _libgcc_mutex             0.1                       main
5    anaconda-client           1.7.2                     py36_0
6    anaconda-project           0.8.3                      py_0
7    ...
```

Install a specific version of a package (I)

```
1    conda install foo-lib=13 # only sepecify MAJOR version
2    conda install foo-lib=12.3 # sepecify MAJOR and MINOR version
3    conda install foo-lib=14.3.2 # sepecify MAJOR, MINOR and
     PATCH
```

Install a specific version of a package (II)

```
  # install 1.0, 1.4 or 1.4.1b2
1conda install 'bar-lib=1.0|1.4*'
2
3# install later than version 1.3.4,
4or earlier than version 1.1
5conda install 'bar-
  lib>=1.3.4,<1.1'
1          (base) $ conda install 'attrs>16,<17.3'
2          Collecting package metadata (current_repodata.json): done
3          Solving environment: failedCollecting package metadata
4          (repodata.json): done
5          Solving environment: done
6
```

```
7           ## Package Plan ##
8
9             environment location: /home/repl/miniconda
10
11            added / updated specs:
12              - attrs[version='>16,<17.3']
13
14
15          The following packages will be downloaded:
16
17              package                    |            build
18              ---------------------------|-----------------
19              attrs-17.2.0               |   py36h8d46266_0         34 KB
20              ------------------------------------------------------------
                                           Total:          34 KB
```

Update a conda package

```
1    conda update PKGNAME
2    conda update foo bar blob
3
4    (base) $ conda update pandas
```

Remove a conda package

```
1    conda remove PKGNAME
```

Search for available package versions

```
1    conda search PKGNAME
```

Find dependencies for a package version

```
1     conda search 'PKGNAME=1.13.1=py36*' --info
2
3
4     (base) $ conda search 'numpy=1.13.1=py36*' --info
5     ...
6     ...
7     dependencies:
8       - libgcc-ng >=7.2.0
9       - libgfortran-ng >=7.2.0,<8.0a0
10      - python >=3.6,<3.7.0a0
11      - mkl >=2018.0.0,<2019.0a0
12      - blas * mkl
```

---

2. Utilizing Channels

Channels and why are they needed?

All Conda packages we've seen so far were published on the main or default channel of Anaconda Cloud. A *Conda channel* is an identifier of a path (e.g., as in a web address) from which Conda packages can be obtained.
Channels are a means for a user to publish packages independently.

Searching within channels
conda search -c channel_name –platform linux-64 package_name

-c, –channel: search in channel
–override-channels: used to prevent searching on default channels
–platform: is used to select a platform

```
1   (base) $ conda search --channel davidmertz --override-channels --platform linux
2   Loading channels: done
3   # Name                  Version           Build  Channel
4   accelerate               2.2.0     np110py27_2  davidmertz
5   accelerate               2.2.0     np110py35_2  davidmertz
6   accelerate-dldist            0.1     np110py27_1  davidmertz
7   ...
8   textadapter              2.0.0         py36_0  davidmertz
9
10
11  (base) $ conda search -c conda-forge -c sseefeld -c gbrener --platform win-64 te
12  Loading channels: done
13  # Name                  Version           Build  Channel
14  textadapter              2.0.0         py27_0  conda-forge
15  textadapter              2.0.0         py27_0  sseefeld
16  textadapter              2.0.0 py27h0ff66c2_1000  conda-forge
17  ...
18  textadapter              2.0.0         py36_0  sseefeld
```

Searching package: anaconda search pck_name
anaconda, not conda

```
1(base) $ anaconda search textadapter
2Using Anaconda Cloud api site https://api.anaconda.orgRun 'anaconda show <USER
3Packages:     Name                  |  Version | Package Types   | Platforms
4    ----------------------- |   ------ | --------------- | ---------------     DavidMertz/textadap
5    conda-forge/textadapter   |    2.0.0 | conda           | linux-64, win-32, osx-64, win-6
6                              : python interface Amazon S3, and large data files     sseef
7                              : python interface Amazon S3, and large data files     stuar
8Found 5 packages
```

conda-forge channel

The default channel on Anaconda Cloud is curated by Anaconda Inc., but another channel called conda-forge also has a special status. This channel does not operate any differently than other channels, whether those others are associated with an individual or organization, but it acts as a kind of "community curation" of relatively well-vetted packages.

```
(base) $ conda search -c conda-forge | grep conda-forge | wc -l
87113
```

About 90,000 packages in conda-forge channel.

Installing from a channel

```
conda install --channel my-organization the-package
```

---

3. Working with Environments

Environments and why are they needed?

Conda *environments* allow multiple incompatible versions of the same (software) package to coexist on your system. An *environment* is simply a file path containing a collection of mutually compatible packages. By isolating distinct versions of a given package (and their dependencies) in distinct environments, those versions are all available to work on particular projects or tasks.

Conda environments allow for flexible version management of packages.

Which environment am I using?

```
(course-project) $ conda env list
# conda environments:
#
_tmp                  /.conda/envs/_tmp
course-env              /.conda/envs/course-env
course-project        *  /.conda/envs/course-project
pd-2015                 /.conda/envs/pd-2015
py1.0                 /.conda/envs/py1.0
test-env               /.conda/envs/test-env
base                  /home/repl/miniconda
```

What packages are installed in an environment? (I)

```
(base) $ conda list 'numpy|pandas'
# packages in environment at /home/repl/miniconda:
## Name                    Version                   Build  Channel
numpy                     1.16.0           py36h7e9f1db_1
numpy-base                1.16.0           py36hde5b4d6_1
```

```
6    pandas                0.22.0          py36hf484d3e_0
```

What packages are installed in an environment? (II)

conda list

-n, –name: env_name

```
1    (base) $ conda list -n pd-2015 'numpy|pandas'
2    # packages in environment at /.conda/envs/pd-2015:
3    ## Name                    Version                   Build  Channel
4    numpy                     1.16.4           py36h7e9f1db_0
5    numpy-base                1.16.4           py36hde5b4d6_0
6    pandas                    0.22.0           py36hf484d3e_0
```

Switch between environments

To *activate* an environment, you simply use conda activate ENVNAME.

To *deactivate* an environment, you use conda deactivate, which returns you to the root/base environment.

```
1     (base) $ conda activate course-env
2     (course-env) $ conda activate pd-2015
3     (pd-2015) $ conda deactivate
4     (course-env) $ conda env list
5     # conda environments:
6     #
7     _tmp                  /.conda/envs/_tmp
8     course-env             *  /.conda/envs/course-env
9     course-project            /.conda/envs/course-project
10    pd-2015                  /.conda/envs/pd-2015
11    py1.0                 /.conda/envs/py1.0
12    test-env               /.conda/envs/test-env
13    base                 /home/repl/miniconda
```

Remove an environment

conda env remove –name ENVNAME

-n, –name

```
1    (base) $ conda env remove -n deprecated
2
3    Remove all packages in environment /.conda/envs/deprecated:
```

Create a new environment

conda create –name recent-pd python=3.6 pandas=0.22 scipy statsmodels

```
1    (base) $ conda create -n conda-essentials attrs=19.1.0 cytoolz
2    Collecting package metadata (current_repodata.json): done
3    Solving environment: done
```

```
4    ...
```

Export an environment

conda env export

-n, –name: export an environment other than the active one

-f, –file: output the environment specification to a file

By convention, the name environment.yml is used for environment, but any name can be used (but the extension .yml is strongly encouraged).

```
1    (base) $ conda env export -n course-env -f course-env.yml
2
3    (base) $ head course-env.yml
4    name: course-env
5    channels:
6      - defaults
7    dependencies:
8      - _libgcc_mutex=0.1=main
9      - blas=1.0=mkl
10     - ca-certificates=2019.5.15=0
11     - certifi=2019.6.16=py36_0
12     - intel-openmp=2019.4=243
13     - libedit=3.1.20181209=hc058e9b_0
```

Create an environment from a shared specification

conda env create -n env_name -f file-name.yml

```
1    (base) $ conda env create --file environment.yml
2    Collecting package metadata (repodata.json): done
3    Solving environment: done
1    (base) $ cat shared-config.yml
2    name: functional-data
3    channels:
4      - defaults
5    dependencies:
6      - python=3
7      - cytoolz
8      - attrs
9
10   (base) $ conda env create -f shared-config.yml
```

---

4. Case Study on Using Environments

Compatibility with different versions

A common case for using environments is in developing scripts or Jupyter notebooks that rely on particular software versions for their functionality. Over time, the underlying tools might change, making updating the scripts worthwhile. Being able to switch between environments with different versions of the underlying packages installed makes this development process much easier.

```
(base) $ cat weekly_humidity.py# weekly_humidity.py
# rolling mean of humidity
import pandas as pd
df = pd.read_csv('pittsburgh2015_celsius.csv')
humidity = df['Mean Humidity']
print(pd.rolling_mean(humidity, 7).tail(5))

(base) $ python weekly_humidity.py
weekly_humidity.py:6: FutureWarning: pd.rolling_mean is deprecated for Serie
        Series.rolling(window=7,center=False).mean()
  print(pd.rolling_mean(humidity, 7).tail(5))
360    77.000000361    80.428571362    78.857143
363    78.285714
364    78.714286Name: Mean Humidity, dtype: float64

(base) $ conda activate pd-2015

(pd-2015) $ python weekly_humidity.py
360    77.000000
361    80.428571
362    78.857143
363    78.285714
364    78.714286
Name: Mean Humidity, dtype: float64
```

FutureWarning is not present in pd-2015 environment.

Updating a script

Update the script so the FutureWarning is gone.

```
(base) $ nano weekly_humidity.py

(base) $ cat weekly_humidity.py
# weekly_humidity.py
# rolling mean of humidity
```

```
import pandas as pd
df = pd.read_csv('pittsburgh2015_celsius.csv')
humidity = df['Mean Humidity']
print(humidity.rolling(7).mean().tail(5))

(base) $ python weekly_humidity.py
360    77.000000
361    80.428571
362    78.857143
363    78.285714
364    78.714286
Name: Mean Humidity, dtype: float64

(base) $ conda activate pd-2015
(pd-2015) $ python weekly_humidity.py
360    77.000000
361    80.428571
362    78.857143
363    78.285714
364    78.714286
Name: Mean Humidity, dtype: float64
```