Using Pixi: A Quick Start Guide

An introduction to Pixi. This guide will help you understand how Pixi can be used and what makes it special

by Sharon Fitzpatrick

What is Pixi?

Pixi is a package management tool

- 🧪 **Conda**: Leverage the existing conda ecosystem to obtain packages written in Python, C, C++, and many other languages.
- 🔄 **Reproducibility**: Work in dedicated, isolated environments that can be easily recreated.
- 🛠️ **Tasks**: Manage complex pipelines effortlessly.
- 🌐 **Multi Platform**: Ensure compatibility across Linux, macOS, Windows, and more.
- 🧩 **Multi Environment**: Compose multiple environments within a single Pixi manifest.
- 🏗️ **Building**: Build packages from source using powerful build backends.
- 📦 **Distributing**: Distribute your software via conda channels or various other options.
- 🐍 **Python**: Full support for `pyproject.toml` and PyPI dependencies.
- 🌍 **Global Tools**: Install globally available tools, safely stored in separate environments.

Key Features of Pixi

🔑 Reliable Environment

Pixi will create your environment the same way every time. Pixi uses a lock file to lock in all your dependencies, so they install the same every time

⭐Pixi Environments can mix conda-forge and pypi dependencies

⚡ Super Fast Dependency Solver

Unlike Conda, mamba, and pip solvers Pixi can solve your dependencies SUPER fast.

Wait so... what exactly is Pixi?

Pixi is what we wanted conda to be.

- Pixi environments can use a mix of conda-forge and pip dependencies natively
- Pixi lets you specify EXACTLY what packages should be install from conda and from PyPi
- Pixi LOCKs the EXACT version of each dependency for each OS
- ❌ No more hoping the environment will solve correctly

Core Pixi Components

This is what makes up a Pixi workspace

🔑 Pixi.lock

lock-file that describes the exact dependencies

📂 pyproject.toml

file that describes the workspace

📦 .pixi

folder that contains the environment

🔑 Pixi still uses conda and pip packages to create your environment

Install Pixi with Powershell

Open Powershell

Execute Installation Command

```
powershell -ExecutionPolicy ByPass -c "irm -useb https://pixi.sh/install.ps1 | iex"
```

Verify Installation

- If you got an error don't worry we address this in the next slide

```
pixi --version
```

Give Pixi Access to Powershell

Option #1: Give Pixi Permission to Run

```
Set-ExecutionPolicy RemoteSigned -Scope CurrentUser
```

Option #2: Temporarily Give Pixi permission to run

- this command is temporary so you will need to re-run it each time you run `pixi shell` in a new powershell window

```
function Invoke-Pixi {
    powershell.exe -ExecutionPolicy Bypass -Command "pixi $args"
}

Set-Alias pixi Invoke-Pixi -Option AllScope
```

Pixi Demo

We will be using Pixi on a new code base to understand how to use its core features

1. Add conda depdencies
2. Add pip depdencies
3. Activate a pixi environment
4. Use Multiple Environments

1. Create the Pixi environment

For this demo we will start by creating a folder called pixi demo and creating our pyproject.toml file with the following command

Initialize the project

```
pixi init —format pyproject
```

- This creates a `pyproject.toml` file and a `pixi.lock` file that will hold the pixi configuration information

```
Windows PowerShell
PS C:\development\7_demos> mkdir pixi_demo
PS C:\development\7_demos> cd .\pixi_demo\
PS C:\development\7_demos\pixi_demo> pixi init --format
pyproject
Created C:\development\7_demos\pixi_demo\pyproject.toml
PS C:\development\7_demos\pixi_demo> ls
    Directory: C:\development\7_demos\pixi_demo
Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        3/27/2025   1:42 PM                src
-a----        3/27/2025   1:42 PM            122 .gitattributes
-a----        3/27/2025   1:42 PM             38 .gitignore
-a----        3/27/2025   1:42 PM            403 pyproject.toml
```

2. Add conda-forge dependencies

Add a conda forge dependency

```
pixi add mkdocs
```

- Adds mkdocs as a conda forge dependency
- Updates the dependencies pyproject file list in the pyproject.toml file

powershell

```
> pixi add mkdocs
Added mkdocs >=1.6.1,<2
>
```

2. Add conda-forge dependencies

Pyproject.toml

- After running `pixi add mkdocs` this updates the `tool.pixi.dependencies` to include mkdocs
- You could have specified a version by running `pixi add mkdocs<1.6.2`

Pyproject.toml

pyproject.toml

```toml
[project]
authors = [{name = "Sharon Fitzpatrick"}]
name = "pixi_demo"
requires-python = ">= 3.11"
version = "0.1.0"
dependencies = []

[build-system]
build-backend = "hatchling.build"
requires = ["hatchling"]

[tool.pixi.project]
channels = ["conda-forge"]
platforms = ["win-64"]

[tool.pixi.pypi-dependencies]
pixi_demo = { path = ".", editable = true }

[tool.pixi.tasks]

# conda forge dependencies
[tool.pixi.dependencies]
mkdocs = ">=1.6.1,<2"
```

3. Add pypi dependencies

Add a pypi dependency

```
pixi add pandas --pypi
```

`-- pypi:` adds pandas a pypi depdency

- Updates the `dependencies` pyproject file list in the pyproject.toml file

powershell

```
PS C:\development\7_demos\pixi_demo> pixi add pandas --pypi
Added pandas >=2.2.3, <3
Added these as pypi-dependencies.
PS C:\development\7_demos\pixi_demo>
```

3. Add PyPi dependencies

Pyproject.toml

```
pixi add pandas --pypi
```

- Updated [project] section to add pandas to the dependencies list
- dependencies: contains all the dependencies from pip

Pyproject.toml

pyproject.toml

```toml
[project]
authors = [{name = "Sharon Fitzpatrick"}]
name = "pixi_demo"
requires-python = ">= 3.11"
version = "0.1.0"
dependencies = ["pandas>=2.2.3,<3"]

[build-system]
build-backend = "hatchling.build"
requires = ["hatchling"]

[tool.pixi.project]
channels = ["conda-forge"]
platforms = ["win-64"]

[tool.pixi.pypi-dependencies]
pixi_demo = { path = ".", editable = true }

[tool.pixi.tasks]

# conda forge dependencies
[tool.pixi.dependencies]
mkdocs = "<2"
```

4. Install and Activate the Environment

Install the Environment

This installs the environment from the pixi.lock file

```
> pixi install
The default environment has been installed.
```

Activate the Environment

- This activates the default environment we made
- equivalent to `conda activate <ENVIRONMENT NAME>`

```
> pixi shell
```

powershell

```
> pixi install
The default environment has been installed.
> pixi shell
(pixi_demo) >
```

⭐ Notice that (pixi demo) is the name of the environment matches the name of the name in the pyproject.toml file

5. Test Pixi Environment

❗ Unlike conda, Pixi environments are specific to the folder they are created in.

Pixi Shell

Now that `pixi shell` is active we can use the environment just like we would with conda.

Exit the Shell

To exit pixi shell enter `exit`

```
(pixi_demo) > exit
>
```

- After you exit the (pixi_demo) is gone

```
> pixi shell
(pixi_demo) > python
>>> import pandas
>>> import mkdocs
>>> quit()
```

Here we open python and we can use the two dependencies we installed and both work!

6. Create Multiple Environments

🌟 Pixi allows you to create multiple environment within the same pixi file

Optional Dependencies

```
[project.optional-dependencies]
geo = ["geopandas"]
```

- `geo` is the name of the environment and `["geopandas"]` is a list of the dependencies for that environment

List all your Pixi Environments

- Under `[tool.pixi.environments]` lists all the environments including the original environment (default) we made earlier

```
[tool.pixi.environments]
default = {features = [], solve-group = "default"}
geo = {features = ["geo"], solve-group = "default"}     # geo environment with geopandas dependency
```

- Solve group tells Pixi to make the environment compatible with the environment listed in the solve-group.
  - Example: `solve-group = "default"` for the test environment means that the "geo" environment will be compatible with the default environment

Pyproject.toml after multi environment

Optional Dependencies

- Lists the two new environments we added

Tool.pixi.environments

- Lists all environments
- Indicates that all environments should be compatible with our default environment

Geo Environment

```
[project.optional-dependencies]
geo = ["geopandas"]

[tool.pixi.environments]
geo = {features = ["geo"], solve-group = "default"}
```

Default Environment

```
[project]
authors = [{name = "Sharon Fitzpatrick"}]
name = "pixi_demo"
requires-python = ">= 3.11"
version = "0.1.0"
dependencies = ["pandas>=2.2.3,<3"]

# conda forge dependencies
[tool.pixi.dependencies]
mkdocs = "<2"
```

pyproject.toml

pyproject.toml

```
4   requires-python = ">= 3.11"
5   version = "0.1.0"
6   dependencies = ["pandas>=2.2.3,<3"]
7
8   [build-system]
9   build-backend = "hatchling.build"
10  requires = ["hatchling"]
11
12  [tool.pixi.project]
13  channels = ["conda-forge"]
14  platforms = ["win-64"]
15
16  [tool.pixi.pypi-dependencies]
17  pixi_demo = { path = ".", editable = true }
18
19  [tool.pixi.tasks]
20
21  # conda forge dependencies
22  [tool.pixi.dependencies]
23  mkdocs = "<2"
24
25  [project.optional-dependencies]
26  geo = ["geopandas"]
27
28  [tool.pixi.environments]
29  default = {features = [], solve-group = "default"}
30  geo = {features = ["geo"], solve-group = "default"}
```

7. Using Multiple Environments

Install New Environment

- `-e :` this means install the environment

```
pixi install -e geo
```

Using the New Environment

```
pixi shell -e geo
```

Switch Environments

- To exit the geo environment and use the default environment
- 🍧 Whenever you run pixi shell pixi will automatically call pixi install behind the scenes

```
exit
pixi shell
```

```
> pixi install -e geo
 The geo environment has been installed.
> pixi shell -e geo
(pixi_demo:geo) > python -c "import geopandas"
(pixi_demo:geo) > exit
> pixi shell
(pixi_demo) >
```

CoastSeg Demo with Pixi

In this demo we will show how to set up CoastSeg with Pixi

1. Install the CoastSeg default environment
2. Launch a notebook
3. Switch Environments to the "ml" environment
4. Launch the zoo notebook in the ml environment

Full tutorial on using CoastSeg with Pixi

Install CoastSeg with Pixi

Open Terminal

Change directories to CoastSeg

```
cd <COASTSEG LOCATION>
```

Run Installation Command

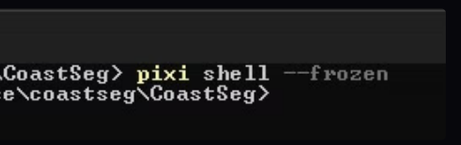
Installs the default environment

- `--frozen` : means install the environment exactly as it is defined in the pixi.lock

```
pixi install --frozen
```

Activate the default environment

```
pixi shell --frozen
```



Troubleshooting Pixi Errors

```
PS E:\anaconda3\CoastSeg-Clone\CoastSeg> pixi shell --frozen
File C:\Users\Redme\AppData\Local\Temp\.tmpdpRhnp.ps1 cannot be loaded because running scripts is disabled on this
system. For more information, see about_Execution_Policies at https:/go.microsoft.com/fwlink/?LinkID=135170.
    + CategoryInfo          : SecurityError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : UnauthorizedAccess
PS E:\anaconda3\CoastSeg-Clone\CoastSeg>
```

- tells powershell that Pixi is safe to connect to Powershell

```
function Invoke-Pixi {
    powershell.exe -ExecutionPolicy Bypass -Command "pixi $args"
}

Set-Alias pixi Invoke-Pixi -Option AllScope

pixi shell --frozen
```

Launch Jupyter Notebook

Verify the installation worked

```
python -c "import coastseg"
```

Launch the notebook

```
jupyter lab SDS_coastsat_classifier.ipynb
```

Terminal

```
C:/CoastSeg > pixi install --frozen
C:/CoastSeg > pixi shell --frozen
(coastseg) C:/CoastSeg >
(coastseg) C:/CoastSeg > python -c "import coastseg"
(coastseg) C:/CoastSeg >
(coastseg) C:/CoastSeg > jupyter lab
SDS_coastsat_classifier.ipynb
```

Switch to ML environment

⭐ Normally tensorflow and transformers are NOT compatible with the rest of the coastseg dependencies but by putting them in their own environment we can use them

Switch Environments

👆 Remember Pixi Shell will automatically run `pixi install` if the environment has not been installed

```
pixi shell -e ml --frozen
```

Import ML dependencies in new environment

```
python -c "import tensorflow; from transformers import TFSegformerForSemanticSegmentation;"
```

Terminal

```
C:/CoastSeg > pixi install --frozen
C:/CoastSeg > pixi shell --frozen
(coastseg) C:/CoastSeg >
(coastseg) C:/CoastSeg > python -c "import coastseg"
(coastseg) C:/CoastSeg >
(coastseg) C:/CoastSeg > jupyter lab
SDS_coastsat_classifier.ipynb
```

Bonus Troubleshooting Tips

#1: Delete the .pixi file to clear existing environment

- This removes all the installed packages
- Run pixi install after removing to re-install all packages
- If things are really bad
  - 1. Delete pixi.lock
  - 2. Delete .pixi
  - 3. Run pixi install

#2: View the dependencies of your packages

pixi tree -i <PACKAGE NAME>

```
PS C:\development\7_demos\pixi_demo> pixi tree -i pandas
pandas 2.2.3
└── pixi_demo 0.1.0
```

pixi tree

```
Windows PowerShell
PS C:\development\7_demos\pixi_demo> pixi tree
├── mkdocs 1.6.1
│   ├── click 8.1.8
│   │   ├── __win
│   │   ├── colorama 0.4.6
│   │   │   └── python 3.13.2
│   │   │       ├── bzip2 1.0.8
│   │   │       │   ├── ucrt 10.0.22621.0
│   │   │       │   ├── vc 14.3
│   │   │       │   │   └── vc14_runtime 14.42.34438
│   │   │       │   │       └── ucrt 10.0.22621.0 (*)
│   │   │       │   └── vc14_runtime 14.42.34438 (*)
│   │   │       ├── libexpat 2.6.4
│   │   │       │   ├── ucrt 10.0.22621.0 (*)
│   │   │       │   ├── vc 14.3 (*)
│   │   │       │   └── vc14_runtime 14.42.34438 (*)
│   │   │       ├── libffi 3.4.6
│   │   │       │   ├── ucrt 10.0.22621.0 (*)
│   │   │       │   ├── vc 14.3 (*)
│   │   │       │   └── vc14_runtime 14.42.34438 (*)
│   │   │       ├── liblzma 5.6.4
│   │   │       │   ├── ucrt 10.0.22621.0 (*)
│   │   │       │   ├── vc 14.3 (*)
│   │   │       │   └── vc14_runtime 14.42.34438 (*)
│   │   │       ├── libmpdec 4.0.0
│   │   │       │   ├── ucrt 10.0.22621.0 (*)
│   │   │       │   ├── vc 14.3 (*)
│   │   │       │   └── vc14_runtime 14.42.34438 (*)
│   │   │       ├── libsqlite 3.49.1
│   │   │       │   ├── ucrt 10.0.22621.0 (*)
```

Bonus Troubleshooting Tips

#3: Use --frozen instead of --locked

- If your lock file is not up to date with your workspace use — frozen to use the dependencies installed in the .pixi folder

```
root@d0ce32be11f6:/coastseg# pixi shell --locked
Error:   x lock-file not up-to-date with the workspace

root@d0ce32be11f6:/coastseg# pixi shell --frozen

(coastseg) root@d0ce32be11f6:/coastseg#
```

Pixi Command Reference Table

Command Reference Table

Command	Description	Conda Equivalent	Documentation
`pixi shell -e <NAME>`	Activate Pixi environment named `<NAME>`	`conda activate <NAME>`	Pixi shell docs
`exit`	Exit the current Pixi environment	`conda deactivate`	Pixi exit docs
`pixi install`	Install dependencies from `pyproject.toml` and update `pixi.lock`	`conda install`	Pixi install docs
`pixi install --frozen`	Install dependencies strictly from `pixi.lock` without updating it, even if it differs from `pyproject.toml`	Install from a conda-lock file	Pixi frozen install docs