Python Project Configure, code and test

Using a specific python version for your python project (pyenv and pipenv)

|  |  |  |
| --- | --- | --- |
|  |  |  |
| .env | make sure this file is copied to root directory which contain key information to access “artifactory” resource in Pipfile |  |
| Install PipEnv – specific version for your project | pyenv local 3.10.11  pyenv exec pip install pipenv --user |  |
| Establish Virtual Environment | Virtualenv location: C:\Users\brusyj\.virtualenvs\  pyenv exec pipenv sync  or  pyenv exec pipenv install |  |
| Install Dependencies | pyenv exec pipenv sync --dev  or  pyenv exec pipenv install --dev |  |
| Active Virtual Environment | pyenv exec pipenv shell  >deactivate  (deactivate: for out of virtual environment ) |  |
| UNDE PyCharm  In ->File  ->Settings  Project: ${project}  Python Interpreter | Python Interpreter: $(project-path)\venv\Scripts\python.exe  Or  Virtual Environment Path  C:\Users\brusyj\.virtualenvs\  C:\Users\brusyj\.virtualenvs\python-glue-large-document-extraction-for--jXwky8sV\Scripts\python |  |
| UNDE PyCharm  In ->File  ->Settings  Project: ${project}  Project Structure | src, test mark as Sources  venv Mark as Excluded |  |
| Code Tests | pytest --cov=src tests/  pytest --cov-report html --cov=src tests/ |  |
| Code Analytic Checks | mypy src/ --ignore-missing-imports |  |
| Code Checks | pylint src/  pyflakes src/  pycodestyle src/ --max-line-length=100  pipenv check |  |
| Code Security Issue Checks | bandit src/ -r -c ./pyproject.toml  or  bandit src/ -r |  |

include:

- project: "DataEngineering/de-tools/gitlab-includes-files"

file: "python/components/sbom\_pages.yaml"

- project: "DataEngineering/de-tools/gitlab-includes-files"

file: "python/components/test\_pipfile\_w\_sbom.yaml"

- project: "DataEngineering/de-tools/gitlab-includes-files"

file: "python/components/sca\_pipfile.yaml"

- project: "DataEngineering/de-tools/gitlab-includes-files"

file: "python/components/deploy\_lambda\_zip\_file.yaml"

**Introduction: pyenv vs pipenv vs virtualenv**

<https://towardsdatascience.com/python-environment-101-1d68bda3094d>

**Pyenv**

Most programming languages have some kinds of version management. Node has nvm and Ruby has RVM, etc. **pyenv** is a Python version management. It installs/uninstalls different Python version, sets global and local (directory) Python version

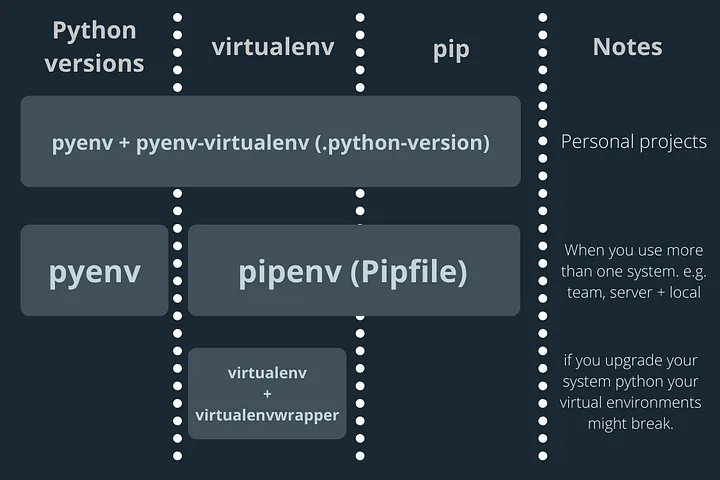
Pyenv is a simple python version management tool. It lets you easily switch between multiple versions of Python. It’s simple, unobtrusive, and follows the UNIX tradition of Sigle-purpose tools that do one thing well (from https://pyenv-win.github.io/pyenv-win/)

**Pipenv**

Pipenv is a packaging tool for Python application and manages package dependencies and its sub-dependencies.

Pipenv simplifies the development process by using a Pipfile in the development environment and a Pipfile.lock in the production environment. Pipenv also manages virtual environments.

**Virtualenv**



Pipenv creates isolated pip environments whereas pyenv+virtualenv creates virtualenvs that multiple projects can use

**Installing pyenv**

<https://pyenv-win.github.io/pyenv-win/>

1. Install pyenv-win in PowerShell.

Invoke-WebRequest -UseBasicParsing -Uri "https://raw.githubusercontent.com/pyenv-win/pyenv-win/master/pyenv-win/install-pyenv-win.ps1" -OutFile "./install-pyenv-win.ps1"; &"./install-pyenv-win.ps1"

1. Reopen PowerShell
2. Run >pyenv –version to check if the installation was successful
3. Run >pyenv install -l, to check a list of Python versions supported by pyenv-win

Yuan: We can use 3.11.5 , 3.10.11 or 3.9.13

1. Run >pyenv install <version> to install the supported version

Installed 3.11.5 , 3.10.11 or 3.9.13

1. Run pyenv global <version> to set a Python version as the global version

> pyenv global 3.11.5

1. Check which Python version you are using and its path

>pyenv version

8. Check that Python is working

>python -c “import sys; print(sys.executable)” \path\to\.pyenv\pyenv-win\versions\3.11.3\python.exe

## pyenv-win commands

commands List all available pyenv commands

local Set or show the local application-specific Python version

global Set or show the global Python version

shell Set or show the shell-specific Python version

install Install 1 or more versions of Python

uninstall Uninstall 1 or more versions of Python

update Update the cached version DB

rehash Rehash pyenv shims (run this after switching Python versions)

vname Show the current Python version

version Show the current Python version and its origin

version-name Show the current Python version

versions List all Python versions available to pyenv

exec Runs an executable by first preparing PATH so that the selected

Python version's `bin' directory is at the front

which Display the full path to an executable

whence List all Python versions that contain the given executable

DELETE the following from System Path on 1/19/2024

C:\Users\brusyj\Anaconda3

C:\Users\brusyj\AppData\Local\Programs\Python\Python311

C:\Python311\Scripts

C:\Python311

**pyenv local**

pyenv local creates or modifies .python-version in the directory. For example,

pyenv local 3.10.11 creates .python-version with Python 3.10.11,

and pyenv 3.11.5 modofies .python-version to Python 3.11.5

Show Python version

> cat .python-version

> python –version

**How to create Virtual environments**

If you are working with your personal projects and not installing pipenv, I recommend installing

pyenv-virtualenv

If you are working in team or with more than one system, I recommend you to install pipenv which I am covering next

**How to create Virtual environments with pipenv**

If your project depends on Python Package versions and its dependencies, pipenv is for you.

Pipenv creates a Pipfile.lock and you use it when you move to a different system.

The Pipfile.lock contains all the dependencies and its versions, and hash.

When you install Python packages using Pipfile.lock, it will create exactly the same environment as your original system.

When a repo has a Pipfile.lock, you can install all the packages and dependencies with pipenv install

Set python version

> pyenv local 3.10.11

Install pipenv

> pip install pipenv

or

> pyenv exec pip install pipenv

Install Packages

> pipenv install –dev

Or

> pyenv exec pipenv install –dev

Create a virtual environment for an exiting project

> pipenv sync –dev

Or

> pyenv exec pipenv sync –dev

Above command produced the following:

Virtuallenv location: C:\Users\brusyj\.virtualenvs\batch-zip-job-gMqyaCej

Installed dependencies from Pipfile.lock

NOTE:

Once you install a package, you can find the package and hashes under default in the Pipfile.lock. This will secure the identifcal environment in a different system

To activate this project’s virtualenv, run pipenv shell

> pipenv shell

Or

> pyenv exec pipenv shell

pipenv Quick Reference

|  |  |
| --- | --- |
| pipenv check | Checks for security vulnerabilitie |
| pipenv --pytho 3.9 | Create a new project using Python 3.9 |
| pipenv --rm | Remove project virtualenv from current directory |
| pipenv install --dev | Insall all dependencies including dev |
| pipenv lock --pre | Create a lockfile containing pre-releases |
| pipenv graph | Show a graph of your installed dependencies |
| pipenv run pip freeze | Use a lower-level pip command |
| pipenv clean | Uninstall all packages not specified in Pipfile.lock |
| pipenv install | Install provided packages and adds them to Pipfile |
| pipenv run | Produces a command installed into the virtualenv |
| pipenv shell | Produces a shell within the virtualenv |
| pipenv sync | Installs all packages specified in Pipfile.loc |
| pipenv uninstall | Uninstalls a provided package and removes it from Pipfile |
| pipenv --where | Locate the project |
| pipenv --venv | Locate the virtualenv |
| pipenv --py | Locate the Python interpreter |

**pipenv works with pyenv**