

# Build-Benedictions

```
$ buildben init-proj
```

## Managing Multiple (Python) Projects & Dependencies





**Dr. rer. nat. Martin Kuric**

---

Academy of Sciences Göttingen · Germania Sacra / HisQu

## `buildben` : Main Commands

( `bube` is the alias for `buildben` )

- `$ bube init-proj` : Create a new **project**. —  99% *Done*
- `$ bube add-experiment` : Add a new **experiment** to a project. —  80% *Done*
- `$ bube env-snapshot` : Dockerize current project for reproducibility. —  80% *Done*
- `$ bube init-database` : Create a new central **database**. —  60% *Done*

# Disclaimer

- `buildben` is very easy to use. (Goal is to make work simpler)
- This presentation is for python beginners.

## But ...

- ... `buildben` solves **a lot** of *behind-the-scenes-problems* at once.
  - The logic behind `buildben` is **not beginner-friendly**.
- Some problems are hard to understand if you haven't encountered them yet...  
(I myself don't understand them fully either, *I simply trust the best practices..!*)
- I'll give my best to explain python standards and my personal decisions.
- If anything is unclear, please ask immediately! (But expect some *(un)organized chaos...*)

## How This all Started:

- I had one big mono-repository containing multiple projects. It was a MESS.
- After splitting into smaller repos: Managing multiple separate projects is painful, too..!
  - "Let's just start developing, I can add a `setup.py` later!"
  - "When did I last update the `requirements.txt` / `setup.py`?"
  - "When anyone tries to use this code, the setup will probably break..."
- I had scripts to automate tasks, all of them poorly documented & scattered across repos!
- I needed one centralized standard to solve **all** my problems:
  - Think ahead, avoid problems, read my mind, etc.
  - Minimal interaction: No more than 1 CLI-command to do 100 things at once.
  - *(like a quick prayer doing miracles ... 🙏🤖)*



Icon of *Jesus Christ Pantokrator* by *Theophanes the Cretan*. His right hand is raised in benediction.

## From Wikipedia:

"A ***benediction*** (Latin: *bene*, 'well' + *dicere*, 'to speak') is a short ***invocation*** for divine help, blessing and guidance [...]."

"***Invocation*** is the act of calling upon a deity, spirit, or supernatural force, typically through prayer, ritual, or ***spoken formula***, to seek guidance, assistance, or presence."

## My Projects before `buildben`:

1. Make a virtual environment ( `.venv` ) for each project:

```
python -m venv ".venv" # Prevents polluting your OS with project-related chaos  
source .venv/bin/activate # Activate virtual environment
```

2. Collect my dependencies in a `"proj-requirements.txt"` file.
3. `pip`: Collects *dependencies of my dependencies* and installs everything:

```
pip install -r "proj-requirements.txt" # Resolve Environment & install dependencies
```

4. Compile all installed dependencies + versions for further reinstalls:

```
pip freeze > "requirements.txt" # Compile list of dependencies installed in current .venv
```

## proj-requirements.txt

- Manually created by me: Whenever I `pip install` a new package, I add it to this file.
- Used by `pip` to *"resolve the environment"* (= collect *dependencies of dependencies*)

```
ipykernel
jupyter    # Convert .ipynb to .py
numpy
openpyxl   # For reading Excel files
pandas
matplotlib
seaborn    # Better plotting
pytest
```

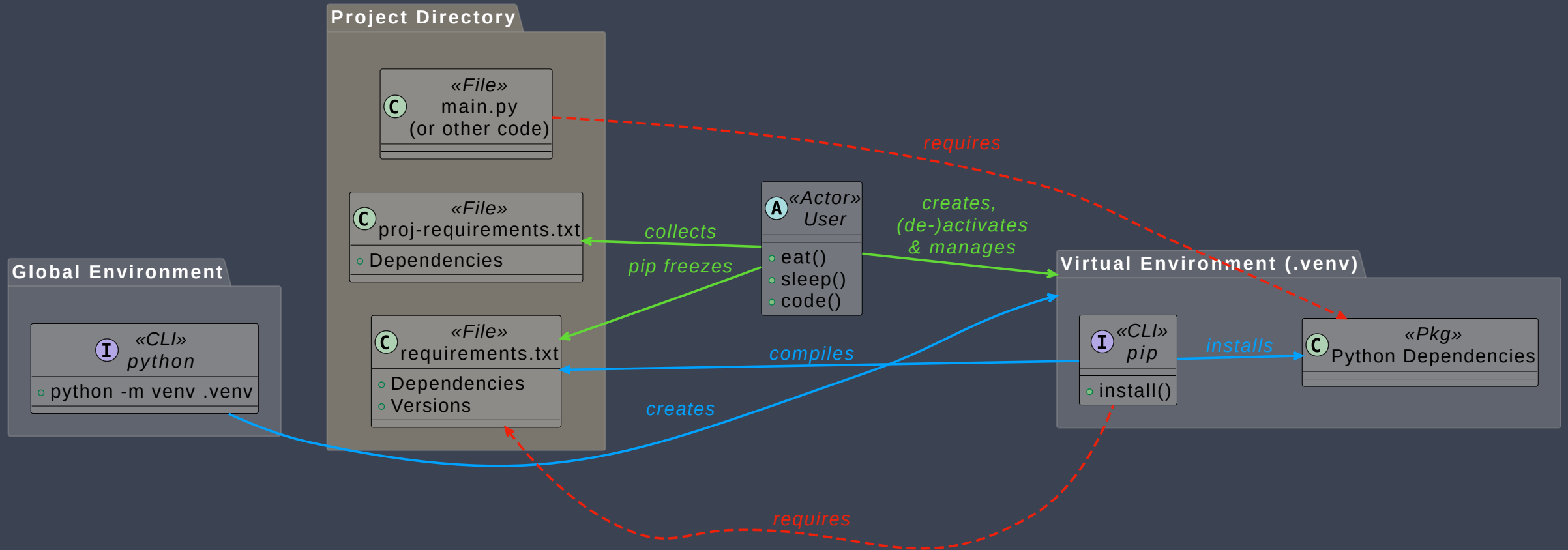
## requirements.txt

```
pip freeze > "requirements.txt" # Compile list of dependencies installed in current .venv
```

```
asttokens==3.0.0
build==1.2.2.post1
click==8.2.1
comm==0.2.2
debugpy==1.8.14
decorator==5.2.1
ipykernel==6.29.5
ipython==9.4.0
ipython_pygments_lexers==1.1.1
jedi==0.19.2
jupyter_client==8.6.3
jupyter_core==5.8.1
matplotlib-inline==0.1.7
# ...
```



# My Projects before **buildben** : Architecture



"I will add a **pyproject.toml** once I need it!"

## My Projects before `buildben` : Setup

```
git clone "<repo-url>"           # Download
cd "<repo-name>"
python -m venv ".venv"           # Prevents polluting your OS with project-related chaos
source .venv/bin/activate        # Activate virtual environment
```

If there's only a `"requirements.txt"` :

```
pip install -r "requirements.txt" # Install only dependencies
```

If there's a `pyproject.toml` :

```
pip install -e .                  # Editable install
```

## My Projects before `buildben` : 2 Main Problems

### 1. Dependencies are pinned by hand:

- `requirements.txt` must be manually updated.

### 2. Imports rely on current working directory:

- `requirements.txt` only holds dependencies, not the **project structure**.
- Cannot import anything outside the current working directory (no `import ../module`)
- VS Code (sometimes) struggles with **refactoring** & **typing** across packages.

## Further Annoyances:

1. `requirements.txt` mixes runtime and development dependencies.
2. (De-)Activating `.venv` can be forgotten or annoying.
3. Too many CLI-commands to remember & type (*especially when working with 4 Repos at the same time*).
4. How to properly write unit-tests mid-development..?

Solution	Why beginners should care	Standard
<code>pyproject.toml</code>	Single file that stores metadata and tool config	<a href="#">PEP 621</a>
Editable install ( <code>pip install -e .</code> )	Code changes are picked up without re-install	<a href="#">PEP 660</a>
<code>src/</code> layout	Forces you to test the installed package	<a href="#">PyPA guide</a>
<code>pip-tools</code>	Auto-generates (and syncs) <code>requirements*.txt</code>	<a href="#">Jazzband docs</a>
<code>direnv</code>	Activates the correct virtual env when you <code>cd</code>	<a href="#">direnv docs</a>
<code>just</code>	Saves “one-liners” like <code>just insco</code>	<a href="#">just README</a>

===== ID-based evaluation =====

Metric	Value
Hits Pos.-ctrl [total count]	2602
Hits AI [total count]	4372
AI TP [count]	1038
AI FP [count]	1105
AI FN [count]	1566
AI UNCLEAR [count]	662
ERROR [count]	1
AI precision [%]	48.4%
AI recall [%]	39.9%

## Build-Benedictions: Minimal Workflow

```
bube proj      # Shorthand for `buildben init-proj`
```

# Build-Benedictions: Improving Setup & Maintenance

## Standardize with template scaffolds:

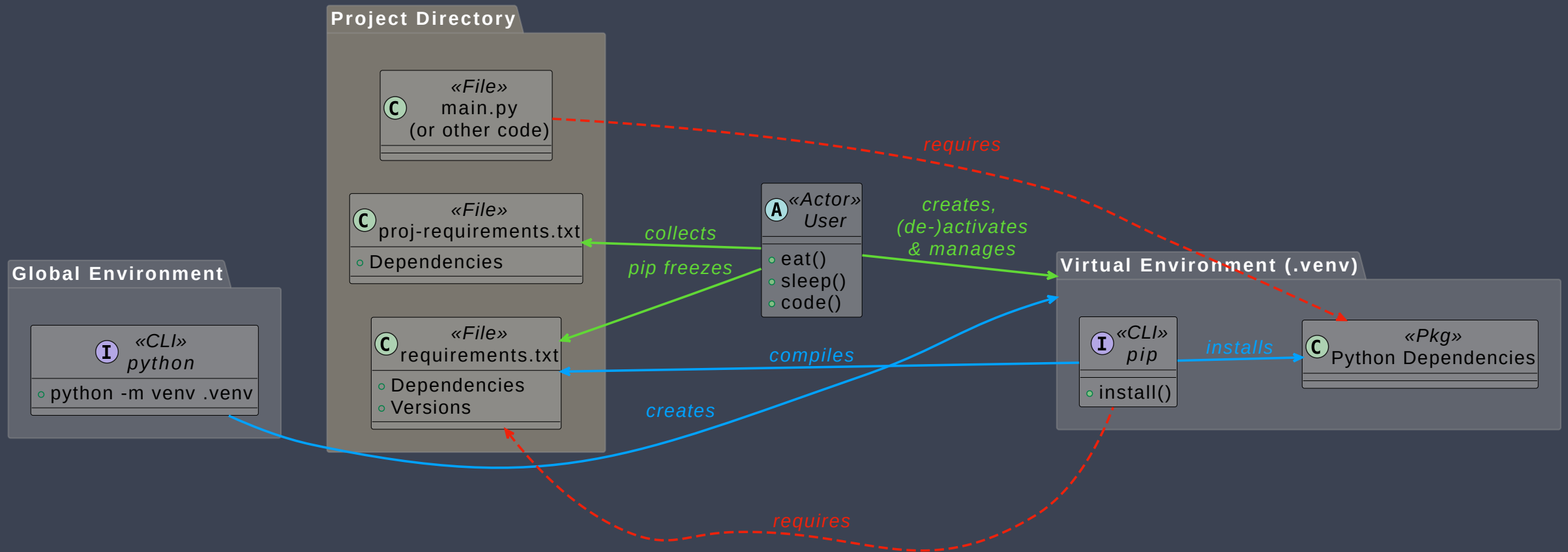
- `$ bube init-proj` : Create a new **project**

## Automate workflow by integrating popular CLI-tools:

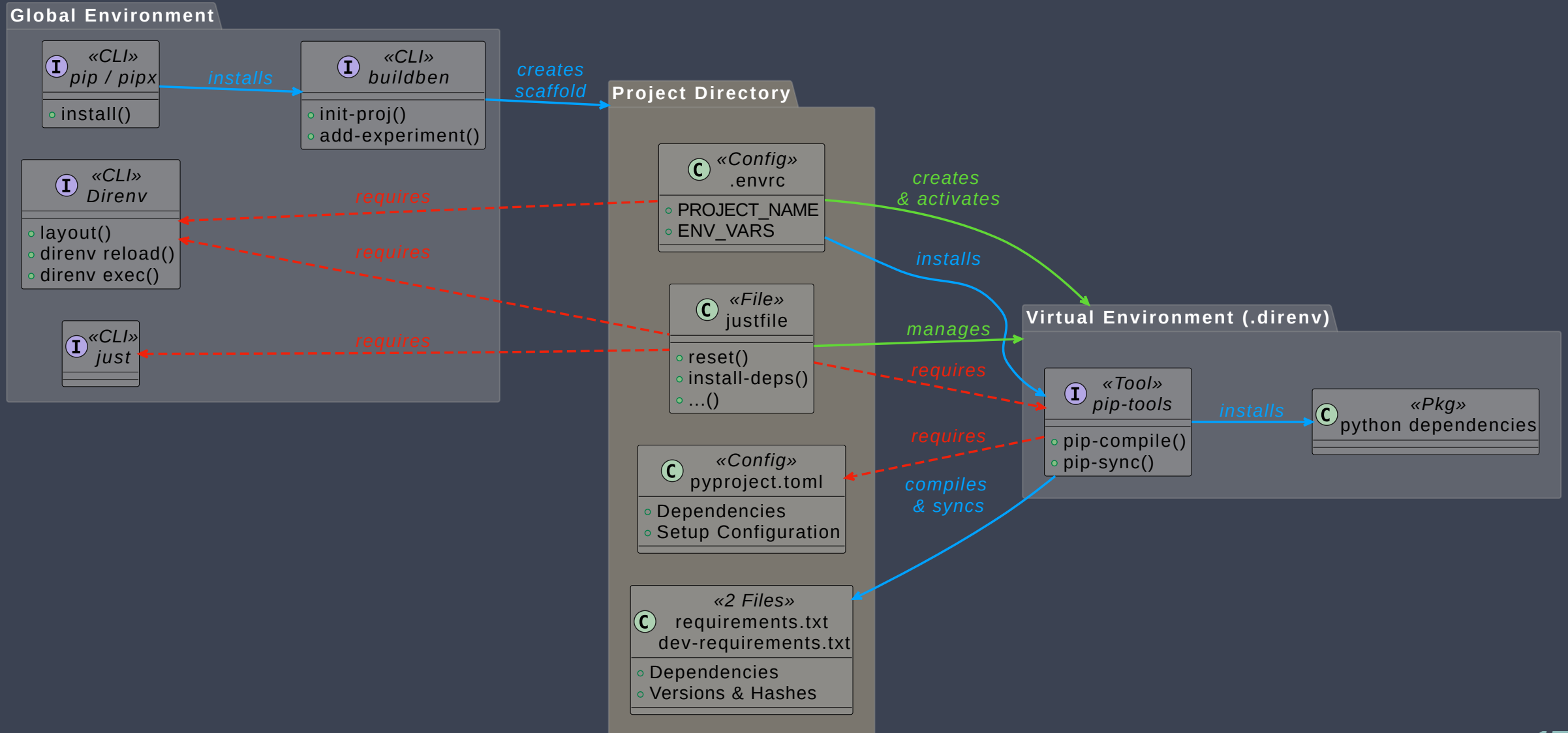
- `direnv` : Automate virtual **environments** & variables.
- `pip-tools` : Automate **dependency** management.
- `just` : Summarize tasks into **one-liners**: `just install`, `just upgrade`, etc.
- `docker` : **Snapshot** current state of project.



# Without buildben :



# With **buildben** :



## Project Directory: `src`-Layout

*# src layout (good)*

```
myproject/  
├── src/  
│   ├── myproject/  
│   │   ├── main.py  
│   │   └── package/module.py  
├── tests/  
│   └── test_module.py  
└── README.md
```

*# flat layout (risky)*

```
myproject/  
├── main.py  
├── package/module.py  
├── tests/  
│   └── test_module.py  
└── README.md
```

### Benefits:

- Avoids imports from working directory via `PYTHONPATH`
  - Forces tests to run on installed code: `pip install -e .` → Catches `import` bugs
- Builds **clean wheels**: Stray files never ship to PyPI
- Recommended by Python Packaging Authority (PyPA)

## Project Directory: Inside **src**

```
myproject/
├── src/
│   └── myproject/
│       ├── __init__.py
│       ├── main.py
│       ├── sheesh.py
│       ├── clients/
│       │   ├── __init__.py
│       │   ├── llm.py
│       │   └── embedding.py
│       └── utils/
│           ├── __init__.py
│           ├── cooltool.py
│           └── module6.py
```

*# Single directory, same name as project root (Recommended)*  
*# Marks directory as package; runs on first import!*  
*# Optional CLI entry-point (wired in via pyproject.toml)*  
*# >>> import myproject.sheesh*  
*# >>> import myproject.clients*  
*# Sub-package "clients"*  
*# >>> import myproject.clients.llm*  
*# >>> import myproject.clients.embedding*  
*# >>> import myproject.utils*  
*# Sub-package "utils"*  
*# >>> import myproject.utils.cooltool*  
*# >>> import myproject.utils.module6*

# Project Directory: Auxiliary Files in Project Root

```
myproject/
├── .venv/           # Virtual environment (or .direnv!)
├── .env             # Environment variables (& secrets)
├── .gitignore
├── .git/            # Repository metadata
├── src/
│   └── myproject/   # Separate source code from tests!
├── tests/
│   └── test_module1.py # Tests for module1
├── justfile         # Development tasks
├── pyproject.toml   # Project metadata, Setup!
├── requirements.txt  # Dependencies
├── requirements-dev.txt # Development dependencies
├── README.md
└── LICENSE
```