Python CLI Project Structure

Command-line application with proper packaging, argument parsing, and distribution.

#python #cli #command-line #typer #click

PNG PDF Copy Prompt

Project Directory

```
mycli/
    pyproject.toml  Project config …
    README.md
    LICENSE
    .gitignore
    .python-version  Pin Python vers…
  src/  Source layout
    mycli/  Package directo…
      __init__.py  Package version
      __main__.py  python -m mycli…
      cli.py  CLI commands an…
    commands/  Subcommand modu…
      __init__.py
      init.py  mycli init
      run.py  mycli run
    core/  Business logic
      __init__.py
      config.py  Config file han…
      runner.py  Core functional…
    utils/
      __init__.py
      console.py  Rich output hel…
      paths.py  Path resolution
  tests/
    __init__.py
    conftest.py  Shared fixtures
    test_cli.py  CLI invocation …
    test_commands/
      __init__.py
      test_init.py
      test_run.py
```

Why This Structure?

This structure uses the `src/` layout recommended by PyPA. It prevents accidental imports from the project root and forces you to install the package to test it—catching packaging issues early. Commands are split into modules for maintainability.

Key Directories

`src/mycli/` - Package code, installed as `mycli`

`commands/` - Each subcommand in its own module

`core/` - Business logic, CLI-agnostic

`__main__.py` - Enables `python -m mycli`

pyproject.toml Setup

```toml
# pyproject.toml
[project]
name = "mycli"
version = "0.1.0"
requires-python = ">=3.10"
dependencies = ["typer>=0.9", "rich>=13"]

[project.scripts]
mycli = "mycli.cli:app"

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

Getting Started

1. Create project with `uv init mycli` or manually
2. `uv add typer rich`
3. Define CLI in `src/mycli/cli.py`
4. `uv run mycli --help`
5. Build with `uv build`

When To Use This

- Building a distributable CLI tool
- Commands with subcommands (like git, docker)
- Need to publish to PyPI
- Want proper `--help` generation
- CLI with config file support

Trade-offs

`src/ layout overhead` - Must install package to run, more setup

`Typer dependency` - Could use stdlib argparse if zero-deps needed

`Commands split` - More files, but scales to many subcommands

Naming Conventions

`Package` - lowercase, underscores ok (my_cli)

`Command files` - Match command name (init.py for `init`)

`CLI entry` - cli.py with `app = typer.Typer()`

Testing Strategy

`CLI tests` - Use `CliRunner` from Typer/Click

`Core tests` - Unit test business logic separately

`Fixtures` - Temp directories, mock configs in conftest.py