perplexity

Cookiecutter is a tool that reads templates and takes actions to set up python projects. I would like a cookiecutter template to : accept the directory in which to work; create the directory if necessary; cd into that directory; use pipenv to create a virtualenv; start the virtualenv; pipenv install django django-environ django-extensions django-easy-logging flake8 black;

Here is a minimal Cookiecutter template that does what you describe using a post-generation hook script.

You'd create a project like this:

```
cookiecutter-django-pipenv/
├── cookiecutter.json
├── {{ cookiecutter.project_slug }}/
│   ├── hooks/
│   │   └── post_gen_project.py
│   └── README.md
```

`cookiecutter.json`

```
{
  "project_name": "My Django Project",
  "project_slug": "my_django_project",
  "work_dir": ".",
  "create_virtualenv": "y",
  "_copy_without_render": [
    "hooks/*"
  ]
}
```

- `work_dir` is the directory you want to work in (relative or absolute).
- `create_virtualenv` lets you skip the pipenv bit if you ever want to.

`{{ cookiecutter.project_slug }}/hooks/post_gen_project.py`

```python
import os
import subprocess
import sys
from pathlib import Path

PROJECT_SLUG = "{{ cookiecutter.project_slug }}"
WORK_DIR = "{{ cookiecutter.work_dir }}"
CREATE_VENV = "{{ cookiecutter.create_virtualenv }}".lower() in ("y", "yes", "true", "1")

def run(cmd, cwd=None):
    print(f"==> {cmd}")
    try:
        subprocess.check_call(cmd, shell=True, cwd=cwd)
    except subprocess.CalledProcessError as e:
        print(f"Command failed with exit code {e.returncode}: {cmd}")
        sys.exit(e.returncode)

def main():
    # Resolve working directory
    work_dir = Path(WORK_DIR).expanduser()
    if not work_dir.is_absolute():
        work_dir = Path.cwd().joinpath(work_dir)

    # Create directory if necessary
    work_dir.mkdir(parents=True, exist_ok=True)
    print(f"Using work directory: {work_dir}")

    # Move generated project into work_dir if we aren't already there
    project_root = Path.cwd()
    target_root = work_dir.joinpath(PROJECT_SLUG)

    if project_root != target_root:
        # Ensure target_root parent exists
        target_root.parent.mkdir(parents=True, exist_ok=True)
        print(f"Moving project from {project_root} to {target_root}")
        # On the filesystem, move the entire tree
        import shutil
        shutil.move(str(project_root), str(target_root))
        project_root = target_root

    # Change into the working directory / project directory
    os.chdir(project_root)
    print(f"Changed directory to: {project_root}")

    if not CREATE_VENV:
        print("Skipping Pipenv environment creation (create_virtualenv != y).")
        return

    # Initialise pipenv, install dependencies
    # This assumes `pipenv` is on PATH
    run("pipenv --python python")  # or specify a version, e.g. pipenv --python 3.12
    run(
        "pipenv install "
        "django "
        "django-environ "
        "django-extensions "
        "django-easy-logging "
        "flake8 "
        "black"
    )

    print("\nEnvironment ready.")
    print("To activate the virtualenv, run:\n")
    print("  cd {}".format(project_root))
    print("  pipenv shell\n")

if __name__ == "__main__":
    main()
```

`{{ cookiecutter.project_slug }}/README.md`

~~~
# {{ cookiecutter.project_name }}

Generated with Cookiecutter + Pipenv.

After generation, if you enabled virtualenv creation, run:

```bash
cd {{ cookiecutter.work_dir }}/{{ cookiecutter.project_slug }}
pipenv shell
```

~~~

### Usage

From whatever directory you keep your templates in:

```bash
cookiecutter path/to/cookiecutter-django-pipenv
```

You'll be prompted for:

- `project_name`
- `project_slug`
- `work_dir` (e.g. `/home/you/dev/foo` or `../foo`)
- `create_virtualenv`

The hook will:

1. Create `work_dir` if needed.
2. Move the generated project directory into `work_dir`.
3. `cd` (effectively) into that directory for the commands it runs.
4. Run `pipenv --python python`.

5. Run `pipenv install` with the packages you listed.

Note: a hook can’t change the *caller’s* shell directory permanently; it just runs commands in that directory. To “start” the virtualenv in your shell you still need to run `pipenv shell` yourself after generation.

⁂

1. https://learn.microsoft.com/en-us/visualstudio/python/using-python-cookiecutter-templates?view=visualstudio
2. https://www.port.io/glossary/cookiecutter
3. https://www.ianwootten.co.uk/2021/01/07/bootstrapping-python-projects-with-cookiecutter-and-makefiles/
4. https://learn.microsoft.com/en-us/visualstudio/python/using-python-cookiecutter-templates?view=vs-2022
5. https://cookiecutter.readthedocs.io/en/latest/README.html
6. https://github.com/cookiecutter/cookiecutter
7. https://cookiecutter.readthedocs.io/en/stable/tutorials/tutorial1.html
8. https://cookiecutter.readthedocs.io/en/1.7.2/first_steps.html
9. https://github.com/frank1010111/cookiecutter
10. https://www.reddit.com/r/Python/comments/1lcz532/a_modern_python_project_cookiecutter_template/