
지금까지 한 것

지금까지 우리는 다음과 같은 것들을 했다:

1. Git (VCS)로 관리해야 하는 부분과 그렇지 않은 부분을 나누고 이해하기
2. 혼자서 Git 을 사용하여 저장소 만들고 프로젝트 역사 관리하기
3. 브랜치를 만들어서 역사를 분기시키고 통합(merge)하고 관리하기
4. forge 사이트(e.g. Git*ub)를 사용하여 프로젝트를 동기화 하고 PR(pull request)을 통해 협업하기

이것들은 모두 충돌-동기화-통합을 관리하며 그것들을 직면하는 방법이었다. 우리는 이제 적어도 다음과 같은 것들은 할 수 있을 (있어야 할) 것이다:

- 프로젝트에서 VCS 로 관리해야 하는 부분과 그렇지 않은 부분을 나누기
- 서로 소스코드를 공유하여 프로젝트 조직원들이 모두 최신 상태의 코드를 충돌 없이 열람하기

앞으로 해야 할 것

이제 우리는 앞으로 다음과 같은 것들을 할 것이다:

- 기본적인 Python 프로젝트 뼈대 구성
- 프로젝트를 forge 사이트에 동기화
- Python 의 venv 를 이용한 virtual environment 구성
- Flask, Waitress 로 기본적인 예시 사이트 구성
- Docker 의 Dockerfile 을 사용하여 Docker 이미지 구성
- nginx 로 Flask container 를 reverse proxy 하기
- docker-compose 를 사용하여 여러 이미지를 통합적인 사용 구성

Python 프로젝트의 기본 (Python Project Layout)

우리는 기본적으로 https://flask.palletsprojects.com/ 의 Tutorial 에 맞춘 프로젝트를 만들 것이다. 이곳에서는 비록 Python 을 기준으로 설명하고 있긴 하지만 실제로는 대부분의 프로젝트에 해당하는 부분이다.

Python 으로 프로젝트를 진행할 때, 어떤 기능을 어떤 모듈이나 어떤 파일로 모아둘지 정하는 것은 프로젝트의 첫 인상을 결정하는 중요한 요소다. 그 프로젝트의 구조가 알기 쉽고 어려움을 결정하는 첫 번째 단계는 바로 이 프로젝트 디렉토리 구조의 결정에 있다.

프로젝트의 디렉토리 구조는 다음과 같은 것들의 기초가 된다:

- 어떤 함수가 어떤 모듈에 들어갈 것인가?
- 데이터 처리는 어떤 흐름에 따라 이뤄질 것인가?
- 어떤 기능이 어떻게 그룹화 되며 어떻게 격리될 것인가?

Python 기반 프로젝트는 사실 사용하는 툴에 따라서 그 형태가 달라질 수도 있다만, 대략적으로는 이곳의 구조를 따른다고 봐도 무방할 것이다.

특히나 Flask tutorial 은 Python 의 전형적이면서 거의 완전한 프로젝트 구조를 보여주기에 매우 도움이 된다.

참고로 https://github.com/pypa/sampleproject/ 에서도 Python 프로젝트의 기본 형태를 제공한다. Python 프로젝트를 구성할 때 참고하면 좋을 것이다.

예를 들어서 hello 라는 project 를 만든다고 할 때, 추천하는 디렉토리 구조는 다음과 같다:

- hello-project/
  - README
  - LICENCE
  - TODO
  - Makefile
  - ChangeLog
  - pyproject.toml
  - requirements.txt
  - hello/
    - __init__.py
    - core.py
    - helpers.py
  - docs/
    - conf.py
    - index.rst
  - tests/
    - context.py
    - test_basic.py
- test_advanced.py

하나씩 설명해두자.

README, README.md, README.rst, README.org 같은 파일은 프로젝트를 소개하는 글이며, 프로젝트를 처음 접하는 사람이 어떻게 프로젝트를 설치하거나 사용하고, 개발에 참여할 수 있는지 등을 적은, **맨 처음 읽는 것이 가정되어 있는 문서다**. 자세한 사항은 docs 이하에 적는 것이 보통이다만, 적어도 README 는 좀 더 자세한 문서로 연결해주는 목차(TOC)적인 역할을 해야한다.

README 는 원하는 형식을 써도 된다. 형식 없는 plain text 는 README 혹은 README.txt 로 적는 경우가 많고, 그 외에도 형식이 있는 plain text 인 README.org, README.md, README.rst 등의 형식도 자주 사용된다. Gitea, Github 등의 서비스는 자동으로 이러한 README 를 인식하여 프로젝트 루트에 기본 표시해주도록 되어 있다.

plain text 이 아닌 형태는 그다지 추천되지 않는다. 되도록 일반적인 text editor 로 읽을 수 있어야 한다.

README 의 문자 인코딩은 UTF-8 이 추천된다. 가끔 구식 시스템(특히 과거의 Windows)을 지원하기 위해 UTF-8 이외의 인코딩을 된 파일을 병행하여 놔두는 경우도 있다. 그런 경우는 README.sjis.txt, README.euc-kr.txt, README.cp949.txt 같이 적기도 한다.

LICENSE 는 프로그램을 대중에 공개하고자 한다면 꼭 있어야 하는 파일이다. 또한 가능하면 모든 소스 파일의 헤더에 라이선스를 명시하는 것이 좋다. LICENSE 는 일반적으로 분리된 파일로 관리하며, 프로젝트의 최상위 폴더에 위치한다.

TODO 는 할 일 목록이다. 있어도 되고 없어도 된다. 작은 프로젝트의 경우 분리하지 않고 README 안에 들어가는 경우도 많다.

ChangeLog 는 프로그램의 버전별 변경 사항을 적어둔 파일이다. 버전이 중요한 프로그램일 경우 이 파일은 중요할 수 있다. 작은 프로젝트의 경우는 README 에 들어가는 경우도 많다.

requirements.txt 는 ¦pip freeze¦ 로 생성한다. 그 대신 Pipenv 를 사용하여 Pipfile 과 Pipfile.lock 을 쓰는 것도 가능하다만, 그 경우도 일반적으로는 requirements.txt 를 관습적으로 놔두는 것이 원칙이다.

pyproject.toml 는 project 의 메타 데이터를 기술하고 빌드 방법이나 빌드 의존성 등을 기술하는 파일이다. 이 파일은 setup.py, setup.cfg 를 대체한다.

setup.py, setup.cfg 는 본래는 Pip 과 Setuptools 이 사용하기 위한, Python 프로젝트의 Makefile 의 위치에 있는 파일이었다. 이는 프로젝트를 하나의 패키지로써 빌드하거나 프로젝트를 설치하거나 할 때 필요한 파일이었다. 다만, 이것은 PEP 517, PEP 518 으로 pyproject.toml 로 대체되었다.

Django 의 경우 manage.py 같은 파일이 있을 수 있는데, 그것도 보통 project root 에 오는 게 적절하다.

Makefile 은 C 프로젝트에서는 전통적으로는 setup.py 의 역할을 맡는 파일이다만, Python 프로젝트에서는 단순한 자동화를 frontend 파일로 사용된다. 문서화 생성, 테스트 실행, 개발 환경 구성 등을 가볍게 적어두면 편리하다.

프로젝트 명을 “hello”라고 한다면, hello/ 는 프로젝트의 프로그램 본체가 들어가 있는 폴더로, hello 라는 모듈명을 가지는 패키지를 의미한다. 만약 hello 라는 프로젝트가 hello.py 라는 단일 스크립트로만 이뤄졌다면 hello 라는 디렉토리 대신 hello.py 하나만으로도 충분하다. (그런 경우는 testing 도 간단할 것이므로 tests/ 대신 tests.py 정도로도 충분할 것이다.)

Python package 디렉토리 안에는 ¦__init__.py¦ 라는 파일이 관습적으로 들어간다. 단순히 hello 를 namespace 로만 쓴다면 빈 파일이라도 문제 없다. 참고로 Python 3.3 이후로는 ¦__init__.py¦ 가 따로 없어도 모듈로 인식이 된다만, 명시적으로 해당 폴더가 Python module 임을 표시하기 위해서 관습적으로 넣어 주는 편이다. 이는 relative import 와도 관련된 문제다.

프로젝트 루트에 그대로 hello/ 혹은 hello.py 가 오는 형태를 flat layout 이라고 하고, src/hello 형태로 오는 경우를 src layout 이라고 한다.

docs/ 에는 문서화 파일이 들어간다. Python 프로젝트의 경우 대부분 Sphinx 를 문서화로 많이 사용하므로, ReST 형식의 plain text 로 많이 구성한다.

tests/ 에는 테스트용 코드가 주로 들어 간다. 프로그래밍에서 테스트는 필수적인 부분이며, 기본적으로는 아무리 많아도 곤란할 게 없다. 작은 프로젝트와 같이 테스트가 그리 많이 필요하지 않은 경우는 tests/ 대신 tests.py 정도의 파일만 있어도 충분하기도 하다.

일반적인 프로젝트

software project 를 구성할 때는 크게 일곱 가지 구성 파일을 가진다. 이 구성은 대략적인 분류이므로 실제 프로젝트에서는 한두 개 정도 빠지거나 더해지는 경우는 있긴 하다만, 대부분은 이 분류에 들어간다.

다음은 그 예시 구성이다.

C 언어 프로젝트의 예시:
- my-c-project/
  - README
  - LICENSE
  - Makefile
  - NEWS
  - src/
    - main.c
    - hello.c
  - include/
    - hello.h
  - test/
  - docs/

Python 프로젝트의 예시:
- my-python-project/
  - README.rst
  - LICENSE
  - Makefile
  - setup.py
  - ChangeLog
  - hello/
    - __init__.py
    - world.py
  - tests/
    - config.py
    - test_world.py
   - ...
- docs/

---

1. readme

readme 는 README, INSTALL, CONTRIBUTING, AUTHORS, MANIFEST, TODO, THANKS 같은 파일을 통틀어서 말하는 것이다. 관습적으로 프로젝트에서 보통 가장 먼저 읽어야 하는 파일이라 인식된다. 프로젝트를 사용하는데 길잡이가 되는 글을 적어두면 된다.

---

2. build code

build code 는 기본적으로는 source code 를 build, deploy, package 하는 등의 명령들을 자동화 하고, test code 를 실행하는 것을 자동화 하는데 사용된다.

Makefile 이 전통적으로 많이 사용되었다만, 프로젝트에 따라서는 CMake, Meson 등의 빌드 시스템을 사용하는 경우도 많다. 프로젝트를 처음 봤을 때 가장 바깥에 나와 있는 make code 에 따라 그 프로젝트를 어떻게 build 해야 할지 알기 쉽도록 각 빌드 프로젝트에 따라서 어느 정도 이름 관습이 있다.

예를 들어서 주로 전통적인 GNU/Linux 의 C/C++ 언어 project 에 사용되는 GNU Autotools 의 경우, Makefile, configure 같은 이름을 가진 파일들로 특정되며, 그 빌드는 ¦./configure; make; make install¦ 이 일종의 숙어처럼 사용된다.

Python 의 경우 setup.py, requirements.txt 같은 이름의 파일로 보통 특정된다. 이는 Python 의 표준적 툴인 Setuptools 와 Pip 을 통해 패키지되어 설치된다.

MANIFEST 같은 이름의 파일도 build code 의 일종이다.

---

3. documentation

문서화는 여러 방법이 있다만, 보통은 어떤 형식을 가진 plain text 로 작성된 것을 문서화를 컴파일 해주는 프로그램을 통해 읽기 좋게 컴파일 하는 형태로 사용된다. 툴에 따라서는 소스 코드를 인식하여 소스 코드의 API 목록을 만들어 주는 툴도 있다.

Java, C, C++ 등의 프로젝트는 Doxygen 이 전통적으로 많이 사용되었고, Python 은 Sphinx 가 주로 사용된다. 또한 Manpage, Info 같은 형식을 사용하는 경우도 있다.

일반적으로 docs 라는 이름의 폴더 이하에 모아진다. 소스 코드 내부에 문서화를 함께 쓰는 경우도 있는데, 프로그램이 커지면 되도록 분리하는 게 간편하다. 애초에 _docstring 과 documentation 은 일단 [[rt:CEKOKDTBLCGE][그 목적이 다르므로]]_ 주의하라. 간단한 프로그램이라면 docstring 정도로도 충분할 수도 있지만, 프로그램이 복잡해지면 문서화가 필수적이게 된다.

---

4. test code

테스트 코드는 프로그램이 의도한대로 동작하는지 확인하기 위한 코드다.

test code 는 build code 를 통해 자동으로 간단히 실행할 수 있어야 한다.

폴더 이름은 보통 tests 나 test 정도다.

---

5. license

license 는 꽤 중요한 파일이다. 프로그램을 공개하고자 한다면 꼭 있어야 한다. 또한 가능하면 모든 소스 파일의 헤더에 라이선스를 명시하는 것이 좋다.

파일 이름은 보통 LICENSE, COPYING 등이다.

---

6. log

log 는 프로젝트의 버전에 따른 변경 사항 등을 적는다. 보통 파일 이름은 LOG, NEWS, ChangeLog 이다.

---
7. source code

소스 코드는 소프트웨어 프로젝트에서는 가장 필수적인 부분이다. 이게 없으면 알맹이가 없는 것이다. 소프트웨어 프로젝트는 이 소스 코드를 감싸는 형태로 구성된다.

단, 소스 코드는 테스트 코드와는 구별되는 것이 일반적이다. 소스 코드 내에 테스트 코드를 함께 적는 경우도 있다만, 프로젝트가 커지면 테스트 코드는 되도록 분리하는 편이 더 간단할 것이다.

소스 코드는 전통적으로는 src 라는 이름의 폴더 안에 들어간다. Python 의 경우 패키지명을 소스 코드의 폴더명으로 쓰는 경우가 많다. (이것 자체는 빌드에서 따로 설정 가능하다)

직접적인 소스 코드 외에도 프로그램에 필요한 구성품이나 간단한 스크립트 종류들도 소스 코드의 일종으로 볼 수 있다. 그런 것들은 asset, scripts, extras 같은 폴더에 보통 들어가게 된다.

예시 코드도 소스 코드의 일부다. 보통 build code 는 예시 코드를 컴파일 하는 명령도 함께 적어둔다. 예시 코드는 examples 같은 폴더에 들어간다.

Python 의 venv 를 이용한 virtual environment 구성

Flask, Waitress 로 기본적인 예시 사이트 구성

Docker 의 Dockerfile 을 사용하여 Docker 이미지 구성

---
nginx 로 Flask container 를 reverse proxy 하기

---
docker-compose 를 사용하여 여러 이미지를 통합적인 사용 구성