Flask app structure, packaging and shipping

What's the product?

What ships, what builds?

Composing the product

What ships?

Often there are (at least) two packages:

- A package for developers/maintainers/extenders, typically called a “source distribution”
- A package for end users and others who will not extend the product, typically called a “binary distribution” (even if it’s not binary)

A service (e.g., a web app) may combine the two

A ‘source distribution’ may be in the form of a version control system repository (e.g., a git repo), like our syllabus app

What goes in the source distribution?

What we wrote

- Code
- Documentation
- Build scripts
- Test suites
- ... anything else needed to continue the project

But what about: libraries, external services, editors and compilers, ... usually not

External dependence and variation

We don't want to ship the Python interpreter with our code ...

but how can we be sure where the Python interpreter is?

Wherever we depend on something we don't ship, we consider what might vary from environment to environment

Adjusting to environment is *configuration*

In our Syllabus app:

CONFIG.py is adjustment of Python variables to environment

pyvenv adjusts the Python interpreter environment

Factoring out the configuration

We don't say: "edit lines 324 and 643 of the Python code to set the appropriate port and directory for your system"

We separate fixed and variable parts, so that it's easier to locate and adjust the variable parts (like CONFIG.py)

We can't keep CONFIG.py under version control (why?)

So we ship CONFIG.base.py with instructions for making CONFIG.py

(This is a common approach, but there are others)

Recurring theme: factor variable parts from fixed parts

Virtual environment: isolating dependence

Table Of Contents

Installation

Flask depends on two external libraries, Werkzeug and Jinja2. Werkzeug is a toolkit for WSGI, the standard Python interface between web applications and a variety of servers for both development and deployment. Jinja2 renders templates.

So how do you get all that on your computer quickly? There are many ways you could do that, but the most kick-ass method is virtualenv, so let's have a look at that first.

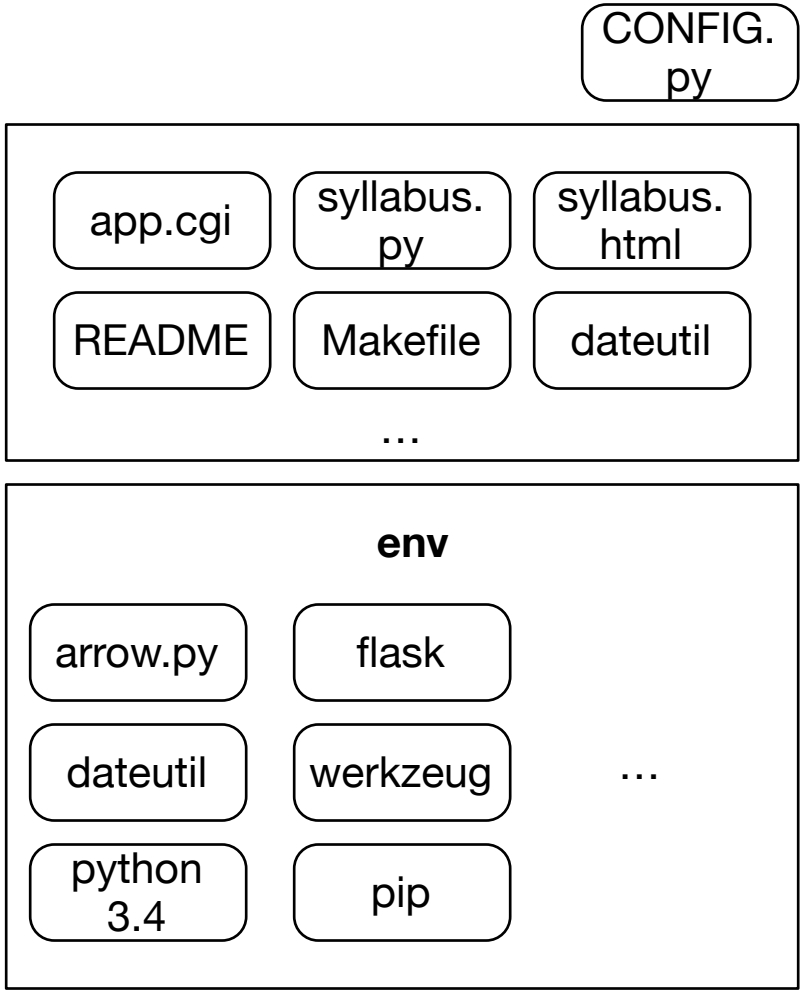
You will need Python 2.6 or higher to get started, so be sure to have an up-to-date Python 2.x installation. For using Flask with Python 3 have a look at Python 3 Support.

virtualenv

Virtualenv is probably what you want to use during development, and if you have shell access to your production machines, you'll probably want to use it there, too.

What problem does virtualenv solve? If you like Python as much as I do, chances are

Virtualenv separates the dependencies of *this* project from the dependencies of *other* projects, so each project can have its own configuration.



Manual configuration

We ship

We download and configure

The Unix path

Unix *environment variables* include a *search path*:

When you type 'python' or 'grep' or 'make', the shell (command line interpreter) looks on $PATH to find the program

$ echo $PATH

/Library/Frameworks/Python.framework/Versions/3.4/bin:/Library/Frameworks/Python.framework/Versions/2.7/bin:/Library/Frameworks/Python.framework/Versions/3.3xc/bin:/Users/michal/Tools/aspectj1.7/bin:/opt/local/bin:/opt/local/sbin: *... (more) ...*

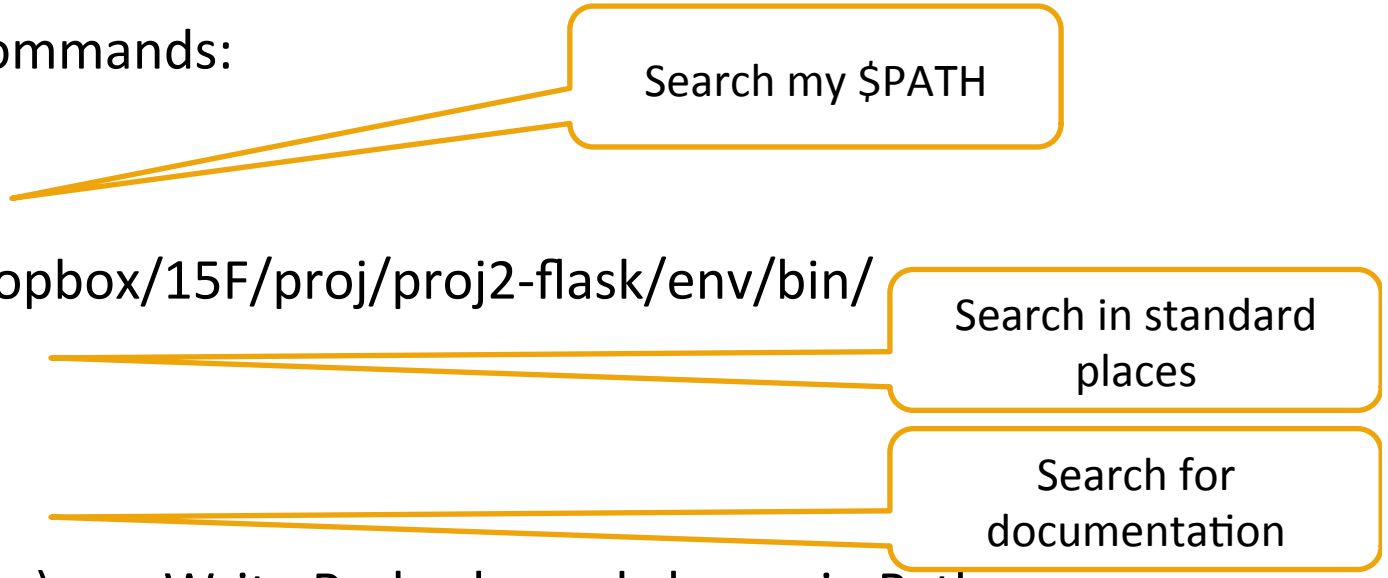
$. env/bin/activate

$ echo $PATH

/Users/michal/Dropbox/15F/proj/proj2-flask/env/bin:/Library/Frameworks/Python.framework/Versions/3.4/bin:/Library/Frameworks/Python.framework/Versions/2.7/bin:/Library/Frameworks/Python.framework/Versions/3.3xc/bin:/Users/michal/Tools/aspectj1.7/bin:/opt/local/bin:/opt/local/sbin: *...(more)...*

Where is that program?

Additional Unix commands:



$ which python

/Users/michal/Dropbox/15F/proj/proj2-flask/env/bin/

$ whereis python

/usr/bin/python

$ man -k python

Inline::Python(3pm) - Write Perl subs and classes in Python

pydoc(1) - the Python documentation tool

python(1) - an interpreted, interactive, object-oriented programming language

python(1), pythonw(1) - an interpreted, interactive, object-oriented programming language

pythonw(1) - run python script allowing GUI

pip and requirements.txt

Instead of including Python libraries in our repo, we include a 'requirements.txt' file for installing the same libraries in another installation

pip freeze: Create requirements.txt from libraries in this env

pip install –r requirements.txt: Recreate libraries from requirements.txt

Recurring pattern: Instead of shipping something, we can ship a recipe for making it. Recipes can be smaller and they can adjust to the local environment.

Flask application pieces

The application source: syllabus.py

Static files: CSS goes here

Templates: HTML templates to be completed with 'session' data (yet another programming language, Jinja2)

Why CSS?

We separate

- Content and document structure in HTML
- Presentation (visual appearance, some behavior) in CSS

Why?

So they can vary independently

- Same content, presented in different ways (e.g., on screen or in print)
- Same style applied to different pages

Recurring theme: Factor things that can vary independently

Let's look at the project

We need to add dates to weeks

arrow is a good package for that.(trust me)

How do we proceed?