A CLOUD GURU

Lesson Description - Planning Through Documentation

To start out our project, we're going to set up our source control, our virtualenv, and finally start documenting how we want the project to work.

Creating the Repo and Virtualenv

Since we're building a project that will likely be more than a single file, we're going to create a full project complete with source control and dependencies. We'll start by creating the directory to hold our project, and we're going to place this in a `code` directory:

```
$ rm ~/requirements.txt
$ mkdir -p ~/code/pgbackup
$ cd ~/code/pgbackup
```

We've talked about pip and virtualenvs, and how they allow us to manage our dependency versions. For a development project, we will leverage a new tool to manage our project's virtualenv and install dependencies. This tool is called `pipenv`. Let's install `pipenv` for our user and create a Python 3 virtualenv for our project:

```
$ pip3.6 install --user pipenv
$ pipenv --python $(which python3.6)
```

Rather than creating a `requirements.txt` file for us, pipenv has created a `Pipfile` that it will use to store virtualenv and dependency information. To activate our new virtualenv, we use the command `pipenv shell`, and to deactivate it we use `exit` instead of `deactivate`.

Next, let's set up `git` as our source control management tool by initializing our repository. We'll also add a `.gitignore` file from GitHub so that we don't later track files that we don't mean to.

```
$ git init
$ curl https://raw.githubusercontent.com/github/gitignore/master/
Python.gitignore -o .gitignore
```

Sketch out the README.rst

One great way to start planning out a project is to start by documenting it from the top level. This is the documentation that we would give to someone who wanted to know how to use the tool but didn't care about creating the tool. This approach is sometimes called "README Driven Development". Whenever we write documentation in a Python project, we should be using reStructuredText. We use this specific markup format because there are tools in the Python ecosystem that can read this text and render documentation in a standardized way. Here's our `READEME.rst` file:

~/code/pgbackup/README.rst

```
pgbackup
========

CLI for backing up remote PostgreSQL databases locally or to AWS S3.

Preparing for Development
-------------------------

1. Ensure ``pip`` and ``pipenv`` are installed
2. Clone repository: ``git clone git@github.com:example/pgbackup``
3. ``cd`` into repository
4. Fetch development dependencies ``make install``
5. Activate virtualenv: ``pipenv shell``

Usage
-----

Pass in a full database URL, the storage driver, and destination.

S3 Example w/ bucket name:

::

    $ pgbackup postgres://bob@example.com:5432/db_one --driver s3
backups

Local Example w/ local path:

::
```

```
    $ pgbackup postgres://bob@example.com:5432/db_one --driver
local /var/local/db_one/backups

Running Tests
-------------

Run tests locally using ``make`` if virtualenv is active:

::

    $ make

If virtualenv isn't active then use:

::

    $ pipenv run make
```

Our Initial Commit

Now that we've created our `README.rst` file to document what we plan on doing with this project, we're in a good position to stage our changes and make our first commit:

```
$ git add --all .
$ git commit -m 'Initial commit'
```