PyNash Coffee and Code Documentation

Release 1.0

PyNash

August 23, 2016

Contents

1 About **3**
- 1.1 How does Coffee and Code work? . . . 3
- 1.2 Does that mean I need to have something prepared to talk about? . . . 4
- 1.3 I hate “public speaking”! . . . 4
- 1.4 Is coffee and code kid friendly? . . . 4

2 Setup **5**

3 Environments **7**
- 3.1 Getting Started . . . 7
- 3.2 pyenv . . . 7
- 3.3 VirtualEnv . . . 9
- 3.4 VIRTUALENVWRAPPER . . . 10
- 3.5 PIP Enhancements . . . 11

4 Indices and tables **13**

Contents:

CHAPTER 1

About

PyNash Coffee and Code (C-n-C) is a "safe learning space." In order to encourage this model, we use the full PyCon Code of Conduct. In addition to that code of conduct we are also using the Hacker School social rules:

- No feigning surprise

  The first rule means you shouldn't act surprised when people say they don't know something. This applies to both technical things ("What?! I can't believe you don't know what the stack is!") and non-technical things ("You don't know who RMS is?!"). Feigning surprise has absolutely no social or educational benefit: When people feign surprise, it's usually to make them feel better about themselves and others feel worse. And even when that's not the intention, it's almost always the effect. As you've probably already guessed, this rule is tightly coupled to our belief in the importance of people feeling comfortable saying "I don't know" and "I don't understand."

- No well-actually's

  A well-actually happens when someone says something that's almost - but not entirely - correct, and you say, "well, actually…" and then give a minor correction. This is especially annoying when the correction has no bearing on the actual conversation. This doesn't mean Hacker School isn't about truth-seeking or that we don't care about being precise. Almost all well-actually's in our experience are about grandstanding, not truth-seeking. (Thanks to Miguel de Icaza for originally coining the term "well-actually.")

- No back-seat driving

  If you overhear people working through a problem, you shouldn't intermittently lob advice across the room. This can lead to the "too many cooks" problem, but more important, it can be rude and disruptive to half-participate in a conversation. This isn't to say you shouldn't help, offer advice, or join conversations. On the contrary, we encourage all those things. Rather, it just means that when you want to help out or work with others, you should fully engage and not just butt in sporadically.

- No subtle 'isms

  Our last social rule bans subtle sexism, racism, homophobia, etc. This one is different from the rest, because it's often not a specific, observable phenomenon ("well-actually's" are easy to spot because they almost always start with the words, "well, actually…").

1.1 How does Coffee and Code work?

Coffee and code is a different kind of meeting than PyNash normally does. Everyone is the teacher and the student! Each C-n-C will be lead by a facilitator, but their main job is to set a framework for the discussion. They aren't there

to lecture the whole time, only to demo and teach first :)

Each person will come up after the facilitator is done and demo/share a part of what they did. This is critical to retention and building confidence/trust.

1.2 Does that mean I need to have something prepared to talk about?

Nope you will demo part of something the facilitator already showed and add anything else you deem helpful.

1.3 I hate "public speaking"!

No worries so do I. But it's a very useful skill much like coding, and this is a safe space.

1.4 Is coffee and code kid friendly?

Yes!!!

CHAPTER 2

Setup

- You need virtualbox installed from here.
- You need vagrant install from here
- Get the Vagrant virtual machine that we'll use to work on from Github

  git clone git@github.com:pynashorg/pynash-cnc.git
- Change into that direction

  cd pynash-cnc
- Next we need to download and start the vagrant box (this will take a while the first time because it has to download an ubuntu cloud image)

  vagrant up
- You're ready to go shutdown the vagrant box and cya Saturday morning!

  vagrant halt

CHAPTER 3

Environments

3.1 Getting Started

Start the VM we'll be working with and connect to it

vagrant up

vagrant ssh

Next, we need to make sure packages are upto date so let's start by refreshing the package list and then upgrading any out of date packages

sudo apt-get update

sudo apt-get upgrade

3.2 pyenv

3.2.1 What is pyenv?

It's a way to build and manage multiple python versions. For example, it will let you run python 2.5, 2.7, 3.4, pypy, etc all happily on one system. It's modeled after rbenv and works well with the rest of what we're going to talk about today.

3.2.2 Setting up pyenv?

First, we need to install git:

sudo apt-get install git

Next we need to clone down the pyenv repo:

git clone git://github.com/yyuu/pyenv.git .pyenv

So that we have access to pyenv we need to hook it up to our shell. In order for this to work properly, we have to tell bash, zsh, or whatever where to look for it.

echo 'export PYENV_ROOT="$HOME/.pyenv"' >> ~/.bashrc

Now we'll tell the shell to add it to the execution PATH list

echo 'export PATH="$PYENV_ROOT/bin:$PATH"' >> ~/.bashrc

Now that we can access the pyenv files, we're gonna call pyenv init

    echo 'eval "$(pyenv init -)"' >> ~/.bashrc

Okay let's restart the shell:

    eval $SHELL

so now we have access to pyenv

    pyenv

Now we need to install the build tools required to compile python sudo apt-get install -y make build-essential libssl-dev zlib1g-dev libbz2-dev libreadline-dev libsqlite3-dev wget curl llvm python-dev

3.2.3 Using pyenv?

Let's see what python we are currently using.

    pyenv versions

    python –version

Let's see what we can install

    pyenv install -l

Let's install the new goodness

    pyenv install 3.4.1

    pyenv rehash

Okay check the versions again

    pyenv versions

Okay let's use python 3.4.1 for our user default python

    pyenv global 3.4.1

    pyenv versions

    python

    import asyncio

Crap back to normal work...

    pyenv install 2.7.8

How can I do per project versions?

    mkdir -p ~/dev/tests

    cd ~/dev/tests

    pyenv local 2.7.8

    ls -al

    cat .python-version

WAIT WAT!?! yeah that's amazing

3.2.4 Extra Goodness

Sets the root for our global version

echo 'export PYVER_ROOT='pyenv prefix'' >> ~/.bashrc

Set the executable path for our global version

echo 'export PYVER_BIN="$PYVER_ROOT/bin"' >> ~/.bashrc

3.3 VirtualEnv

Let's us seperate python packages into convenient environments that we can enable and disable. This lets us do things like deal with dependancies and pin versions.

3.3.1 Install virtualenv

pip install virtualenv

3.3.2 Using virtualenv

virtualenv

WAIT WAT?!?

pyenv which python pyenv which virtualenv pyenv rehash

3.3.3 Creating an environment

virtualenv venv

3.3.4 Activating an environment

source venv/bin/activate

notice the prompt change

cat venv/bin/activate

Now we can install packages in this virtual environment that don't inteerfer with our system python or any other python apps we're working on

Let's install another package

pip install flask

3.3.5 Leaving the virtualenv

which python

pip freeze

deactivate

Notice the python and package listings

    which python

    pip freeze

So what is I don't wanna use the pyenv version of python I want a different one

    virtualenv –python=/opt/python-3.3/bin/python venv

3.4 VIRTUALENVWRAPPER

Makes it easier to setup and use virtualenv in a consistent manner project to project. It also provides some great hooks for us to tie into.

3.4.1 Install virtualenvwrapper

    pip install virtualenvwrapper

Tell virtualenvwrapper where to store virtualenvs

    echo 'export WORKON_HOME=$HOME/.virtualenv' >> ~/.bashrc

Tell virtualenvwrapper where to store projects

    echo 'export PROJECT_HOME=$HOME/dev' >> ~/.bashrc

Initialize virtualenvwrapper

    echo 'source $PYVER_BIN/virtualenvwrapper.sh' >> ~/.bashrc

reinit shell

    source ~/.bashrc

3.4.2 Using virtualenvwrapper

3.4.3 Listing available environments/projects

    workon

3.4.4 Creating an environment

This creates and activates a new virtualenv but does not create a directory

    mkvirtualenv cookies

Deactivating doesn't change it's just

    deactivate

3.4.5 Removing an environment

    rmvirtualenv cookies

3.4.6 Creating a project

This creates a new virtualenv and a project directory.

mkproject cookies

3.4.7 Removing a project is a two step process

rm -rf $PROJECT_HOME/cookies

rmvirtualenv cookies

3.4.8 Activating an environment or project

This will activate the environment and if a project switch to it's directory

workon cookies

3.4.9 Hooks

let you add to the behavior of the virtualenvwrapper commands

cd ~/.virtualenv

ls

An example

3.5 PIP Enhancements

Pip can be so much faster than it is, but it requires just a few things done to it first

Glyph's pip 2014 awesomeness

tweet @glyph a HUGE THANK YOU ... RIGHT NOW from PYNASH!

3.5.1 The pain

pip install ipython[all]

3.5.2 Installing Packages

pip install setuptools;

pip install wheel

pip wheel setuptools

pip wheel virtualenv

pip install virtualenv virtualenvwrapper

3.5.3 Setting up ENV

echo ‘export STANDARD_CACHE_DIR=”${XDG_CACHE_HOME:-${HOME}/.cache}/pip”’ >> ~/.bashrc

echo ‘export WHEELHOUSE=”${STANDARD_CACHE_DIR}/Wheelhouse”’ >> ~/.bashrc

echo ‘export PIP_USE_WHEEL=”yes”’ >> ~/.bashrc

echo ‘export PIP_DOWNLOAD_CACHE=”${STANDARD_CACHE_DIR}/Downloads”’ >> ~/.bashrc

echo ‘export PIP_FIND_LINKS=”file://${WHEELHOUSE}”’ >> ~/.bashrc

echo ‘export PIP_WHEEL_DIR=”${WHEELHOUSE}”’ >> ~/.bashrc

3.5.4 Using it right

pip wheel ipython

pip install ipython

CHAPTER 4

Indices and tables

- genindex
- modindex
- search