Virtstrap Documentation

Release 0.3.1

Reuven V. Gonzales

March 14, 2012

CONTENTS

virtstrap makes your life easier, by providing a simple and repeatable bootstrapping process. It was inspired by pip+virtualenv and buildout, but seeks to create a unified and standard way that won't scare away any new python developers and will make old pythonista's lives a bit easier. All with the wonderful convenience of the MIT License.

With virtstrap, setting up your development environment is as simple as this:

```
$ vstrap init
```

This will make a new virtualenv in a directory `.vs.env` and a file called `quickactivate` in your current working directory. To use this virtualenv just type:

```
$ source quickactivate
```

Python development should be this simple. That's the goal of this project. To make setting up a project for python as simple as developing the project itself.

CHAPTER

ONE

CURRENT FEATURES

- Provides a standard location for virtualenv
- Provide a quick and simple way to activate the current environment
- Generate a requirements file much like a Gemfile.lock

CHAPTER

TWO

FUTURE FEATURES

- Provide a simple plugin system
- Allow for arbitrary environment variables to be set

CHAPTER

THREE

USER GUIDE

3.1 Virtstrap Installation Guide

3.1.1 Recommended method

Pip is the recommended method for installing virtstrap. Simply do:

```
$ pip install virtstrap
```

If you only have easy_install, you should install pip before continuing and execute the command above.

Once it has completed all the correct dependencies should be installed.

3.1.2 Installation from source

Due to the complexity of installing from source. It is highly discouraged at this time. However, this should hopefully work for most people

First grab the code:

```
$ git clone git://github.com/ravenac95/virtstrap-suite.git
```

Install the code:

```
$ make install
```

If you'd like to be able edit the code use this command instead:

```
$ make install-develop
```

But if you're ready to do that you may want to look at *Contributing to virtstrap* to get started.

3.2 Virtstrap Quickstart Guide

First, make sure you have virtstrap installed. If you do not, head on over to the *Virtstrap Installation Guide*.

3.2.1 Simplest virtstrap example

After virtstrap has been installed a command, vstrap, will be available on your command line. You can create an virtstrap enabled project just by doing the following:

```
$ mkdir myproject
$ cd myproject
$ vstrap init
```

This creates a virtualenv in the directory `myproject/.vs.env` and a bash script at `myproject/quickactivate`.

Finally, do:

```
$ source quickactivate
```

You now have a virtualenv for `myproject`.

3.2.2 Virtstrap with basic configuration

In the previous section we created the most simple type of virtstrap environment possible. However, without any configuration files virtstrap is a bit anemic. So let's start a simple configuration file to go along with the previous example.

In your favorite editor start a file called `VEfile` in your myproject directory (mine is vim):

```
$ vim VEfile
```

Let's say you'd like to grab two packages: Armin Ronacher's wonderful Flask micro web framework, and Kenneth Reitz's amazing requests HTTP library. Put the following inside `VEfile`:

```
requirements:
  - flask
  - requests: '>=0.10'
```

Save your file and run this command in your shell:

```
$ vstrap install
```

This command runs the installation portion of the init command. Doing `vstrap init` would have had the same effect. The `install` command skips some of the steps involved in `init`

After the command completes it's work, you will now have the latest version of *flask* and any *requests* package greater than version 0.10 inside your virtual environment. In order for this to happen, virtstrap converted the requirements defined in VEfile to a pip requirements. Pip then takes over and installs all of the requirements.

In the future, the VEfile will also generate a file called VEfile.lock which will contain the exact versions of the packages you just installed. This file like, Ruby Bundler's Gemfile.lock, should be added into your repository to create a truly repeatable project environment.

3.2.3 Repeatable environments. Because it matters

A repeatable environment is the main goal of virtstrap. As such let's take a look at exactly how that all works.

First let's get rid of the virtstrap environment. *VEfile and VEfile.lock are not deleted*:

```
$ vstrap clean
```

This brings an almost bare directory, save the configurations defined in VEfile.lock and VEfile. Finally do:

```
$ vstrap init
```

Your project is now brought us back to the state before we ran `vstrap clean`. The implication of this is that say you and Bob are working on this project together. Instead of emailing you and asking you about all the dependencies or manually creating a virtualenv and running a pip requirements all bob has to do is type the following inside his cloned project directory:

```
$ vstrap init
```

Now you're both ready to go. Beautiful isn't it :-)?

3.3 The VEfile

The VEfile is a central point of virtstrap. It allows you to define project metadata, requirements, and eventually options for plugins. The VEfile is a YAML file that uses some unique conventions to define the configuration.

3.3.1 It's just YAML

To understand the VEfile here's a short introduction to it's structure. The following is a valid VEfile:

```
foo: bar
unladen: swallow
python_is: awesome
```

In it's most basic form the VEfile is a simple dictionary or key/value storage. The top most level of keys are considered "sections" and their values can be anything. In the example above the sections are `foo`, `unladen`, and `python_is`. In python this VEfile simply becomes:

```
{'foo': 'bar', 'unladen': 'swallow', 'python_is': 'awesome' }
```

Just remember, you can define any key/value pair you wish in the VEfile and virtstrap will happily ignore any section (key) it doesn't recognize.

3.3.2 Virtstrap Sections

For the sections virtstrap does recognize, it expects particular types of values (although it's still pretty lenient). By design, none of the sections in virtstrap are required. This allows you to use virtstrap without any real specifications. However, once you're done being lazy and not setting up your project's repeatable environment, here are the sections you can set.

- *project_name*: Defines the project name. By default the project name is inferred from a projects root directory name. Set this if you'd like it to ensure consistency no matter where it is located.
- *requirements*: Defines the requirements for the project. This is the most useful section and one that you will probably use most. Requirements are explained in the next section, *The "requirements" Section*

3.3.3 The "requirements" Section

The requirements section of the VEfile allow you to define your project's dependencies. Currently there are three forms of dependency declaration.

1. *Package name* - This is the simplest declaration. All you do is use the package name so your VEfile would look like this:

```
requirements:
  - some_package # Syntax
  - flask        # Example
```

2. *Package name with version specification* - This declaration allows you to specify a version or a range of versions. The syntax is similar to defining a just a package name, but it separates the specification string from the package name by a colon. See here:

   ```
   requirements:
     - some_package: "some_spec"  # Syntax
     - flask: ">=0.7"             # Example
     - requests: "<=1.0"          # Another example
   ```

   `some_spec` can be any specification that is allowed by python's distutils.

3. *Package name with urls* - This declaration is the most complex and is meant to be used when you'd like to grab a package from a repository. The syntax may seem verbose for those used to pip's requirement syntax, but it is meant to be read more easily and hopefully more usable as well. See here:

   ```
   requirements:
     - some_package2:              # Syntax for normal urls
       - url_to_package_tar_or_zip

     - some_package1:              # Syntax for VCS
       - vcs_type+url_to_repo      # vcs_type must be git|bzr|hg|svn
       - editable: true            # This is optional and makes
                                   # a package editable

     - requests:                   # Example1 (normal url)
       - https://github.com/kennethreitz/requests/tarball/v0.10.6

     - flask:                      # Example2 (VCS url)
       - git+https://github.com/mitsuhiko/flask
       - editable: true
   ```

   Those familiar with pip will see that the syntax isn't too far off. The basic syntax for urls is one of two different types: the VCS url or a normal url. A VCS url **must** be preceded by a type, which is any of the following: git, hg, bzr, or svn. The normal url must point to a tar, zip, or a local directory.

Here's a full example of a requirements section that installs `flask`, `requests`, `virtstrap-core`, and `virtstrap-local`.

```
requirements:
  - flask
  - requests: ">=0.7"
  - virtstrap-core:
    - git+https://github.com/ravenac95/virtstrap-core.git
    - editable: true
  - virtstrap-local:
    - https://github.com/ravenac95/virtstrap-local/tarball/v0.3.0
```

3.3.4 Profiles

One additional, and powerful, part of VEfile's structure is it's ability to use profiles. In virtstrap, a profile is a particular type of environment you'd like to setup. These types of environments could be something like *development*, *testing*, *staging*, *production*, etc. Virtstrap makes little assumptions about the names you with to use for profiles. The *development* profile is the single exception. Virtstrap will always use the *development* profile if you do not specify a

different profile. The reason for this is that most of your time with virtstrap will be spent developing code, so it should be simple.

In order to define profiles, VEfile utilizes YAML's concept of documents. Each document in a YAML file is separated by a `---`. The first document in the VEfile is always the default profile. This profile is always used regardless of the currently chosen profile. Every document after that must define a section `profile` whose value will be used as the profile name. Here's an example of a VEfile that uses profiles:

```
###############################################
# This section is the default profile
# it is ALWAYS used. So don't put anything here
# that isn't absolutely necessary on every
# environment
###############################################
project: tobetterus

requirements:
  - sqlalchemy
  - flask: ">=0.7"

some_value: foo

--- # This starts a new document (therefore a new profile)
#######################################
# This profile is the development profile
# as defined by the section directly
# below this comment
#######################################
profile: development

# Lists and dictionaries always append the other profile's data
# when profiles are combined
requirements:
  - ipython

# If it isn't a list or dictionary it's value
# is overridden entirely.
# So the value of some_value if you use the
# development profile will be 'bar'
some_value: bar

---
profile: production

requirements:
  - python-memcached
  - mysql-python
```

The VEfile above defines 3 profiles: *default*, *development*, and *production*.

To use profiles all you have to do is specify the `--profiles` options on the command line interface. You do this like so:

```
$ vstrap [command] --profiles=production,development
```

The line about will use both the production and the development profile. So the list of requirements installed will be `sqlalchemy`, `flask`, `ipython`, `python-memcached`, and `mysql-python`. In addition, if you request for the value `some_value` you will get the value `bar`, but that's only really useful if you're developing a plugin for virtstrap.

3.3.5 VEfile Suggestions

These are some suggestions when creating a VEfile.

- Use spaces instead of tabs (this is pretty much a suggestion for everything you write).
- Use 2 spaces for each tab level. This makes VEfiles a bit easier to read.
- Try not to specify exact versions for requirements in the VEfile. It is most powerful when you do not do that. Virtstrap is able to lock all the requirement versions so you can repeat your environment on each machine.
- Don't specify absolute file URL's. This makes your project less repeatable.

CHAPTER

FOUR

DEVELOPER GUIDE

4.1 Contributing to virtstrap

In order to provide for the an easy setup for the user, virtstrap has been split into 3 different packages. That are all combined into a single repository, virtstrap.

- **virtstrap** - This is the main package that users see. It provides the console script `vstrap` which is the main interface to anything virtstrap related. It also contains the commands that can be used without the presence of a project.
- **virtstrap-core** - This is the core of all of the virtstrap logic. The majority of virtstrap's code is contained in this core package. It is also a dependency for the other two packages.
- **virtstrap-local** - This package contains any commands that can only be used within a project and not throughout the system.

4.1.1 Start developing!

To start contributing to virtstrap is pretty simple. First, fork the repository on github. Once you've done that do the following:

```
$ make develop
$ source quickactivate.sh
```

Now you'll be in a virtualenv made for virtstrap.

4.1.2 Virtstrap Makefile

The virtstrap repository contains a Makefile that has the following commands:

- `develop` - Setup the development environment using an old version of virtstrap
- `testall` - Runs all of the tests in all the packages
- `supportfiles` - Builds the support files and places them into the virtstrap_support folder inside the virtstrap package.
- `install` - Installs virtstrap and virtstrap-core
- `install-develop` - Installs virtstrap and virtstrap-core as development versions (they're editable)