freckles Documentation

Release 0.1.97

Markus Binsteiner

Aug 14, 2017

Contents

Contents:

CHAPTER 1

freckles

managing dotfiles! and more! cute!

freckles is configuration management for your local, or virtual, machine(s). If you are familiar with ansible, puppet, chef, or **saltstack_**, you know about configuration management, and why it (mostly) is a good idea. If not: in short, configuration management gives you a way to describe a machine/server and the services and applications it runs. Either in code, or a configuration format like json or yaml. Then it takes that configuration and applies it to a machine, removing the need for you to setup the machine maunually, as well as guaranteeing that the machine is always setup the same way, even after a re-install.

Because of the overhead that come with configuration management systems, using them is usually restricted to situations where the infrastructure to be controlled is deemed to cross a certain threshold of... let's call it 'importance'. While for production services, or other business-relevant systems this threshold is often crossed even for single servers, this is not usually the case for the physical (or virtual) machines developers (or somesuch) use when going about whatever they go about. There are exceptions of course, but spending the time to learn about, and then setting up a system like that is not always worth it. *freckles* tries to change that equation by making it easier, and faster, to apply the principles of configuration management to local development environments. I do think there's a lot of developers time to be saved, to be used on actual development, rather than all the annoying stuff around it...

Blahblah. Yes, sorry. Example, to keep you interested:

```
$ curl https://frkl.io | bash -s -- freckles gh:makkus/dotfiles
```

This is what I use to setup a new machine, after a) I buy a new Thinkpad or b) I did something silly that requires a re-install. You figure out what happens more often. Anyway, what this does is:

- bootstraps *freckles* itself, then straight away executes it
- expands the `gh:makkus/freckles` url to https://github.com/makkus/dotfiles (optional to have a short url, but I grew to like those short ones)
- checks out the repository to $HOME/freckles/dotfiles (this is configurable of course)
- reads all the metadata it can find in that repository, describing mostly which packages to install
- installs all the packages it found in the metadata (btw, same metadata can be used to describe the setup on several flavors of Linux as well as on Mac OS X)
- metadata also says that this repository is of type `dotfiles`, so *freckles* goes ahead and symbolically links all the configuration files it finds in the repository into their appropriate place in my home directory (using an application called *stow* – which it also installs if not present already)

My *dotfiles* are a bit more involved that

Here's how the (common part) of the metadata looks like: XXX link. And here is how the `dotfiles` profile works: XXX link

2nd example, executed directly after the above run:

> Now, after setting up my machine with all the apps and configuration I use, I want to start working on *freckles* again, which, I guess I should tell you, is not all that finished or stable just yet. Which I have to start working on *freckles* again, see. Thus:
>
> ```
> $ freckles gh:makkus/freckles
> ```
>
> Here's what happens:
>
> - freckles is already installed, so I can call it directly now (had to login again, or execute `source $HOME/.profile` to pick up the path *freckles* is installed in)
> - as before, expands the url, from `gh:makkus/freckles` to https://github.com/makkus/freckles
> - checks out the repository to $HOME/freckles/freckles
> - reads the metadata, installs the packages that are necessary (virtualenv and pycrypto dependencies, mostly, in this case)
> - also figures out this is a python dev project, so it:
> - creates a virtualenv
> - installs all the requirements it can find (in requirement*.txt files in the root folder of the repo) into the new virtualenv
> - executes `python setup.py develop` within that same virtualenv

Those two are the only so-called *profiles* I have implemented so far: `dotfiles` and `python-dev`. *freckles* is written in a way to add more of those profiles fairly easily though, my reasoning being that its a good idea to have a set of 'commonly used', 'best-practices' profile of how code should be structured, and which metadata is necessary to describe certain 'expressions' of that code or data (e.g. a python project could need to be setup in a development environment, or installed from source for 'normal' use).

I haven't finished thinking about all potential pros and cons yet, but so far I think that metadata should sit with the code itself (with a few minor exceptions like for example where on the target machine it should be checked out). Once that is done, we can have systems do things automatically to get the target system in the state that is determined by the code itself, the profile used, and some aspects of the host machine (e.g. which OS is running on it, which package managers are available).

The nice thing about this is that this gives you all the advantages of an automated system to manage your working space, while still allowing flexibiliy in how to deal with certain types of code/data. For example, you don't like the `stow` way of symbolically linking dotfiles? Well, just create a profile that sets up your dotfiles using a detached git repostory (XXX link). As long as the repository contains the name of the profile in its metadata, all is good.

Right. There's more, but I realize this is already too much text for a project Readme. So instead of writing more text here, I'll write more text elsewhere:

- frecklecute
- freckles profiles
-
- Free software: GNU General Public License v3
- Documentation: https://freckles.readthedocs.io.

Features

- TODO

Credits

mac_pkg: Spencer Gibb (https://github.com/spencergibb/battleschool)

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Bootstrap

There are a few different ways to bootstrap *freckles*. Depending on the state of your box, your proficiency and your general trust in random people on the internet, you can choose one of the methods below.

Bootstrap & execution in one go:

This method of bootstrapping is the easiest, and fricktionlessest. One of the main reasons for creating *freckles* was that I wanted a way to setup a new box (physical or virtual) by executing only one (easy to remember) line on a commandline terminal. This functionality was factored out of *freckles* itself into a project called 'inaugurate'.

In the below examples we'll bootstrap *freckles* using either user, or root permissions, and then execute it directly, calling its help function. When used in anger, you'll probably point it to a *freckle* git repo, and let it do its thing directly, but for the purpose of this we'll not let it make any changes to your system (except for the install itself, obviously).

Here's the (base) command you'll have to execute to bootstrap & execute *freckles*. If you want to execute *freckles*' sister programm *frecklecute* instead, just replace *freckles* with *frecklecute* below and it'll work the same way.

```
curl https://frkl.io | bash -s -- freckles --help
```

Or, if you don't have *curl*, but *wget* installed on your box:

```
wget -O - https://frkl.io | bash -s -- freckles --help
```

Once you executed either one of the above commands successfully, you'll have *freckles* installed on your system. It'll have put a line in either '~/.profile', or if that doesn't exist, '~/.bash_rc' to add its path to the session PATH, so the next time you login (or do a `source ~/.profile`) it'll be available. So, from then on all you need to type is:

```
freckles --help
```

Bootstrap via script (without elevated permissions)

This is the default way of bootstrapping *freckles*. It will create a self-contained installation (in `$HOME/.local/inaugurate/`), using conda to install requirements.

Commands

Using *curl*:

```
curl https://frkl.io | bash -s -- freckles <args>
```

Using *wget*:

```
wget -O - https://frkl.io | bash -s -- freckles <args>
```

Supported

Those are the platforms I have tested so far, others might very well work too:

- Linux
  - Debian
    - Stretch
    - Jessie
  - Ubuntu
    - 17.04
    - 16.10
    - 16.04
  - CentOS
    - 7
- Mac OS X
  - El Capitan
  - Sierra
- Windows
  - Windows 10 (Ubuntu subsystem) – not tested/working yet

What does this do?

This installs the conda package manager (miniconda actually). Then it creates a conda environment called 'inaugurate', into which *freckles* along with its dependencies is installed.

Everything that is installed (about 450mb of stuff) is put into the `$HOME/.local/inaugurate/conda/envs/inaugurate` folder, which can be deleted without affecting anything else (except, of course, you did install some other applications using *conda*, those might be deleted too of course).

If a `$HOME/.profile` file exists, a line will be added to add `$HOME/.local/bin` to the users `$PATH` environment variable. If no such file exists, it will add this line to `$HOME/.bashrc`. If that doesn't exist either, it's the users responsibility to either add that path manually, or start *freckles* directly using its path (`~/.local/bin/freckles`).

Bootstrap via script (with elevated permissions)

This is a quicker way to bootstrap *freckles*, as 'normal' distribution packages are used to install dependencies. Also, the size of the `$HOME/.local/inaugurate` folder will be smaller, ~70mb – systems packages are adding to that in other parts of the system though). The *freckles* install itself is done in a virtualenv using *pip*. Root permissions are required.

Supported

Those are the platforms I have tested so far, others might very well work too:

- Linux
  - Debian
    - Stretch
    - Jessie
  - Ubuntu
    - 17.04
    - 16.10
    - 16.04
  - CentOS
    - 7
- Mac OS X
  - El Capitan
- Windows
  - Windows 10 (Ubuntu subsystem) – not tested/working yet

Using *curl*:

```
curl https://frkl.io | sudo bash
```

Using *wget*:

```
wget -O - https://frkl.io | sudo bash
```

What does this do?

This installs all the requirements that are needed to create a Python virtualenv for *freckles*. What exactly those requirements are differs depending on the OS/Distribution that is used (check the *Install manually via pip* section for details). Then a Python virtual environment is created in `$HOME/.local/inaugurate/virtualenvs/inaugurate` into which *freckles* and all its requirements are installed (~70mb).

If a `$HOME/.profile` file exists, a line will be added to add `$HOME/.local/bin` to the users `$PATH` environment variable. If no such file exists, it will add this line to `$HOME/.bashrc`. If that doesn't exist either, it's the users responsibility to either add that path manually, or start *freckles* directly using its path (`~/.local/bin/freckles`).

Install manually via `pip`

If you prefer to install *freckles* from pypi yourself, you'll have to install a few system packages, mostly to be able to install the `pycrypto` and `cryptography` packages when doing the `pip install`.

Requirements

Ubuntu/Debian

```
apt install build-essential git python-dev python-virtualenv libssl-dev libffi-dev␣
↪stow
```

RedHat/CentOS

```
yum install epel-release wget git python-virtualenv stow openssl-devel stow gcc␣
↪libffi-devel python-devel openssl-devel
```

MacOS X

We need Xcode. Either install it from the app store, or do something like:

```
touch /tmp/.com.apple.dt.CommandLineTools.installondemand.in-progress;
PROD=$(softwareupdate -l |
       grep "\*.*Command Line" |
       head -n 1 | awk -F"*" '{print $2}' |
       sed -e 's/^ *//' |
       tr -d '\n');
softwareupdate -i "$PROD" -v;
```

We also need to manually install pip:

```
sudo easy_install pip
```

Install *freckles*

Ideally, you'll install *freckles* into its own virtualenv. But if you read this you'll (hopefully) know how to do that. Here's how to install it system-wide (which I haven't tested, to be honest, so let me know if that doesn't work)

```
sudo pip install --upgrade pip   # just to make sure
sudo pip install freckles
```

Optionally, if necessary (if you didn't do a systemwide install) add *freckles* to your PATH. for example, add something like the following to your `.profile` file (obviously, use the location you installed *freckles* into, not the one I show here):

```
if [ -e "$HOME/.virtualenvs/freckles/bin" ]; then export PATH="$HOME/.virtualenvs/
↪freckles/bin:$PATH"; fi
```

Bootstrapped files/layout

The bootstrap process will install *freckles* as well as its requirements. *freckles* (and depending on the bootstrap process choosen, also its dependencies) is installed into `$HOME/.local/inaugurate`. Symbolic links `freckles` executable as well as some helper applications (`ansible-playbook`, `conda`, etc.) are created in `$HOME/.local/bin` and a line is added to `$HOME/.profile` or `$HOME/.bashrc` which adds this folder to

the `PATH` variable, which means that after the next login (or after issuing `source ~/.profile`) *freckles* can be run directly from then on.

CHAPTER 3

Commands overview

`freckles` consists of a few different commands, which all use the same underlying codebase and configuration format.

A short side note: `freckles` applications mostly operate on urls. To make those urls more memorable, and also shorter, a few (optional – you can still just provide the full url) abbreviation schemes are supported.

- *github repo*: `gh:<github_user>/<repo_name>`
- *github repo file*: `gh:<github_user>/<repo_name>/path/to/file`
- *bitbucket_repo*: `bb:<bitbucket_user>/<repo_name>`
- *bitbucket_repo_file*: `bb:<bitbucket_user/<repo_name>/path/to/file`

Following is a list of (currently) available commands:

freckles

- freckles usage page
- (currently) supported freckle profiles
- examples

`freckles` is an application that downloads a remote code or data repository (I reckon it's a bit silly to call such a repository a *freckle*, right? too bad...) that is structured according to one or some conventions. After download, `freckles` will execute pre-defined tasks appropriate for the type of *freckle* in question.

For example, if the *freckle* is a python project, `freckles` will create a virtualenv named like the repository (after, if necessary, downloading everything that is needed to create virtualenvs in the first place), download and install all dependencies it can find in any potential `requirement_*.txt` files, and then execute `python setup.py develop`, both inside the created virtualenv.

In fact, this is the recommended way to get started if one wants to contribute to the `freckles` project itself:

```
$ freckles gh:makkus/freckles
```

This will prepare a virtualenv in which development of the `freckles` project can be done.

Or, the *freckle* is a folder containing subfolders which in turn contain *dotfiles* (that's what configuration files are called in Unix-land). `freckles` will download this repo, install potentially configured applications that relate to the configuration files, and symbolically link those configuration files to the appropriate places.

Here is how I initialize a newly installed machine (Linux, Mac, VM, container, doesn't matter) using my dotfiles from https://github.com/makkus/dotfiles (using curl to bootstrap `freckles` itself as documented in the bootstrap page XXX):

```
$ curl https://frkl.io | bash -s -- freckles gh:makkus/dotfiles
```

This one command will automatically download my dotfiles repository onto the machine, then read all the `.dotfiles.freckle` metadata files contained within to find out which applications should be installed, and finally symbolically link (using an application called **stow_** XXX) all the configuration files into their appropriate places in my home directory.

Checkout the above link for a list of currently supported profiles.

frecklecute

- frecklecute usage page
- (currently) officially supported *frecklecutables*
- examples

Where `freckles` automatically applies pre-configured tasks according to a profile, `frecklecute` is more flexible. It takes a `yaml`-formatted text file (while we're assigning silly names, let's call those `frecklecutables`, shall we?) as input, and executes the list of tasks contained in them.

That doesn't sound like much, for sure. Just a script of some sort script, using yaml. There are a few differences to 'normal' shell scripts though, and those might or might not make sense, depending on what needs to be done.

Most of those differences stem from the fact that `freckles` is built as a layer on top of ansible, and uses the rich ecosystem of ansible modules and roles. Ansible is a powerful and rich configuration management system, and if you haven't heard from it, look it up, it is pretty impressive piece of code, in my opinion. Ansibles main purpose is to help setup and maintain compute infrastructure. You write some configuration that describes the setup of your environment, and ansible will make sure that environment is setup that way. More details about configuration management: XXX other page

One such difference is that `frecklecutables` are (for the most part) idempotent. That means, if you run the same command a second time, you can be sure nothing changed on the machine you ran it on, and if it ran successfully the first time, it'll run successfully a second time. A shell script might run successfully one time, but might error out a second time, say, when it tries to create a folder it already created in an earlier run.

Anyway. Long story short, in some situations its good to use configuration management, sometimes its overkill, because of the overhead those systems introduce. This is the reason that configuration management is mainly used for managing infrastructure that crosses a certain threshold of, lets say... importance. Developer laptops or VMs don't often do that, except of course if the developer in question recognizes the importance of configuration management, and has the time and/or expertise to set it up.

So, now. While ansible itself is already quite user-friendly (for a configuration management system anyway), it takes a non-trivial amount of work to execute what is called in ansible terms a *playbook* (a list of tasks, or script if you

will). For starters, ansible itself (and its dependencies) has to be installed. Then an inventory of hosts to manage has to be created (even if it is only used locally), some *ansible roles* (XXX) or other dependencies might have to be downloaded, and put in the right places. Finally you have to write the playbook itself. True, most of those things only need to be done once, and for the rest you can prepare templates. But in case you don't wanna, and you still want to take advantage of all the awesome ansible modules and roles out there, `frecklecute` is for you.

Say, you want to install

EXAMPLE XXX

This will work on

CHAPTER 4

Advanced usage

TBD

CHAPTER 5

Examples

Here I'll collect example configurations for commonly encountered use-cases. I figure it's probably more useful than the usage guides, because I hope the configuration syntax is simple enough to grasp so it'll be easy enough to understand what is going on by having a look at the configuration, without needing too much in the way of explanations.

Quickstart config

This is the configuration used in the *freckles* Quickstart guide, explained in some detail in the 'Usage' page.

Link: Usage

Freckles dev config

This configuration sets up your machine so you can start developing on *freckles* itself.

Link: Freckles development config

Markus' own workstation config

My own configuration, showcasing how to use sub-folder paths within a dotfile repository to split the configuration in several parts which can be mixed and matched.

CHAPTER 6

Trust

freckles bootstrap

So, you think it's sorta bad that all those newfangled projects nowadays want you to download a script with curl or wget and pipe it directly into bash? I agree. It is sorta bad. First and foremost from a security perspective of course, but also for a number of other reasons. *freckles* is not in any way better, and in some regards even worse. For now, I'm not quite sure what else to do though, especially given one of my main objectives, which is to be able to setup a workstation with a one-liner, without having to install *freckles* and it's dependencies manually before executing the first run. Of course I give that option, but I'm not sure how much better that really is, since, if you intend to use *freckles* you have to trust it's code in the first place. So running a bootstrap script from my github page/domain redirect is not really that much worse than trusting my *freckles* package on pip (which is what you'll have to do in either case).

installer scripts used by freckles

Then there is the matter of freckles installing package managers like *nix*, *conda*, or *homebrew*. Some of those are also installed using the recommended method, which is downloading a script with curl or wget and piping it directly into bash. I'm not really too happy doing this, but I don't really

Applying freckles configurations

I'm not sure whether people will end up using *freckles* at all, and how much they like or dislike its configuration format. As I've mentioned somewhere else, that 'elastic' configuration format is an experiment, and it might turn out to be way too involved and unusable for other people to even consider picking it up. If there is any sort of usage from other people than myself though, it might turn out that people share configurations on how to setup certain projects (like the one that sets up a *freckles* dev environment) or working environments

CHAPTER 7

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

Types of Contributions

Report Bugs

Report bugs at https://github.com/makkus/freckles/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

Write Documentation

freckles could always use more documentation, whether as part of the official freckles docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/makkus/freckles/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Get Started!

Ready to contribute? Here's how to set up *freckles* for local development.

1. Fork the *freckles* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/freckles.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv freckles
$ cd freckles/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 freckles tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.

2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/makkus/freckles/pull_requests and make sure that the tests pass for all supported Python versions.

Tips

To run a subset of tests:

```
$ py.test tests.test_freckles
```

CHAPTER 8

Credits

Development Lead

- Markus Binsteiner <makkus@posteo.net>

Contributors

None yet. Why not be the first?

CHAPTER 9

History

0.1.0 (2017-06-02)

- First release on PyPI.

CHAPTER 10

Indices and tables

- genindex
- modindex
- search