freckles Documentation

Release 0.1.70

Markus Binsteiner

May 18, 2017

Contents

Contents:

CHAPTER 1

Overview

managing dotfiles? and more? cute!

At its heart, *freckles* is configuration management for your local machine(s). If you are familiar with ansible, puppet, or chef you know what configuration management is, and why it's a good idea. If not: in short, configuration management gives you a way to describe the configuration of a machine and the services and applications it runs in some way and format (e.g. in code, json, yml, ...) and apply this recorded configuration onto a vanilla (virtual or not) machine.

Depending on the configuration management framework you choose, the learning curve for them is quite steep, and require quite a bit of initial investment in terms of time to learn how they work, and to prepare the configuration that works for your infrastructure. Even the (arguably) easiest to pickup system, *ansible*, needs you to prepare and edit several files and folders, even for simple use-cases. This makes sense, since all of those solutions are quite powerful, and have to be able to deal with quite a lot of complexity.

freckles' goal is to simplify this configuration for use-cases that are less involved, like setting up your local workstation and development VMs with the tools of your choice, and the configuration needed to get started quickly. Without having to do a lot of manual bootstrapping, setup and preparation. Ideally, issuing one command, involving a pre-created configuration should be enough.

freckles is implemented as a layer on top off *ansible*. Instead of describing your infrastructure, as you do in *ansible*, in *freckles* you describe your working environment (in general, or for a specific project). This is only a subtle difference, and I'm still not sure whether its worth developing a project like *freckles*. The idea got me curious enough to try and find out though :-).

freckles is written in Python, and GPL v3 licensed.

Documentation: https://freckles.readthedocs.io.

Features

- one-line setup of a new environment, including freckles bootstrap
- minimal and (hopefully) intuitive config file format, using `yaml` syntax
- supports Linux & MacOS X (and probably the Ubuntu subsystem on Windows 10)
- share the same configuration for your Linux and MacOS workstation as well as Vagrant machines, containers, etc.
- support for systems where you don't have root/sudo access via the nix package manager or conda (or if you just think it's a good idea to use any of them)
- direct support for all ansible modules and roles

Really quick-start

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

This bootstraps *freckles*, runs it, and displays help information. All files that are installed live under the `$HOME/.freckles` folder, which can be deleted without affecting anything else. This also adds a line to your `$HOME/.profile` file to add *freckles* to your path.

Quickstart

Warning: run this only after you read what it does, as it installs some packages onto your computer you might not want. Should not do any real harm though.

For its most basic use-case – which is installing and configuring packages – *freckles* needs:

- one or more *configuration file(s)*
- *curl* (or *wget*) -> for bootstrapping (well, technically it also needs *bash*)
- optionally, a *dotfile repository* -> if some of the applications you want *freckles* to install have configuration files

At the moment (and that might change in the future), the easiest way to install *freckles* is to bootstrap it (more details: *Bootstrap*) using curl and bash. The bootstrap process can optionally also execute the first *freckles* run, which makes it possible to setup a machine with one line in your shell. Like:

```
curl -sL https://get.frkl.io | bash -s -- apply gh:makkus/freckles/examples/
↪quickstart.yml
```

The config file I've choosen as an example is a bit more complicated than it'd need to be, but I wanted to show off how *freckles* can use the same config file for different platforms. If you only work on one platform, the same config would look quite a bit tidier. Check out the same example for (only) Debian/Ubuntu: quickstart-debian.yml.

Either, way, the above command applies the following (fairly) simple configuration to your machine:

```
vars:
  dotfiles:
     - base_dir: ~/dotfiles-quickstart
       remote: https://github.com/makkus/freckles-quickstart.git

tasks:
  - checkout-dotfiles
  - install:
      use_dotfiles: true
      packages:
        - epel-release:
            pkgs:
              yum:
                - epel-release
        - htop
        - fortune:
            pkgs:
              apt:
                - fortunes
                - fortunes-off
                - fortunes-mario
              yum:
                - fortune-mod
```

```
              homebrew:
                - fortune

- stow
- create-folder: ~/.backups/zile
```

What this does:

- checks out the repository of dotfile(s) at https://github.com/makkus/freckles-quickstart.git
- on Mac OS X, installs homebrew if it is not installed already (this does not need to be specified, *freckles* figures that out on its own)
- installs the `epel-release` repo if on a RPM-based platform
- installs all the applications/packages that are configured in the repo we checked out earlier (only the emacs-like editor `zile` in this case) – this is done by setting the `use_dotfiles` variable of the `install` task to true
- also installs a few other packages that don't require configuration which is the reason they are not included in the dotfiles repo (`htop` and, depending on which platform this is run on one or some more packages for the *fortune* tool)
- stows all the dotfiles in the above repository into the users home directory (again, only for *zile* in this case)
- creates a folder `$HOME/.backups/zile` if it doesn't exist already (needed because it is configured in the `.zile` config-file – contained in the repo we checked out and 'stowed' (means symbolic-linked) to the user home directory – to be used as backup directory. *zile* does not create that dir itself and errors out if it doesn't exist)

To read how all that works in more detail, please read the full documentation at: *Usage*

You don't like executing random scripts on the internet? Yeah, me neither. Read here: *Trust*

What, ...why?

I re-installed a new (or recently bricked) laptop or VM or container this one time too often, and I was annoyed that there is no real easy and quick way to re-create my working environment in those fresh environments, without having to write shell-scripts that sooner or later turn out unmaintainable and are fairly unflexible to begin with. Now, of course, that's what configuration management tools are for, and I do quite like ansible and have a bit of experience with it. What I don't like is how one usually needs a set of configuration files to describe a setup, even for simple use-cases like setting up a single, local machine. And I didn't want to install *ansible* itself manually every time before I can run my playbooks and roles. Basically, I wanted a thing that allows me to run one line of code, pointing to one configuration file, and after a while I have the same setup as I have on my other machines.

This is what *freckles* now is, sorta. As a result of my tendency to over-engineer everything in my way along with me having a bit of time on my hands – it now can do a few other things which I didn't consider before I started working on it, and which may or may not be useful to somebody else. Either way. If you want a simple and lightweight script to manage your machine, you better run, fast. But if you don't mind a bit of what angry oldish IT folk and/or minimalism-hipsters would probably call 'bloat', and you think that a bit of harddrive-space is a good trade-off for saving a few minutes/hours every once in a while, give this here a go and tell me what you think.

Supported platforms

Currently tested and supported

- Debian
  - Jessie
- Ubuntu
  - 16.04
  - 16.10

Planned / Partially supported

- MacOS X (should mostly work)
- Windows 10 (Ubuntu on Windows)

License

Credits

This package was created using, amongst others:

- ansible
- Cookiecutter
- nix
- conda
- ansible-nix

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:

Below I only describe bootstrapping *freckles* itself. You can, however, execute *freckles* itself in the same go. If you want to do that, replace `bash` with `bash -s -- <freckles_command>`. For example:

```
curl -sL https://get.frkl.io | bash -s -- apply gh:makkus/freckles/examples/
↪quickstart.yml
```

Bootstrap via script (without elevated permissions)

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

Supported

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

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

Using *curl*:

```
curl -sL https://get.frkl.io | bash
```

Using *wget*:

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

If a `$HOME/.profile` file exists, the bootstrapping process adds a line to add `$HOME/.freckles/bin` to the users PATH, so *freckles* can be started directly in the future. To use *freckles* right after bootstrapping without having to logout and login again, do:

```
source $HOME/.profile
```

Or, if that doesn't apply to your machine because you don't have a `.profile` file, do something like this (how-/wherever you see fit):

```
export PATH=$PATH:$HOME/.freckles/bin
```

What does this do?

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

Everything that is installed (about 450mb of stuff) is put into the `$HOME/.freckles` folder, which can be deleted without affecting anything else.

If a `$HOME/.profile` file exists, a line will be added to add `$HOME/.freckles/bin` to the users `$PATH` environment variable. If no such file exists, it's the users responsibility to either add that path, or start *freckles* directly using its path (`~/.freckles/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/.freckles` folder will be smaller, ~70mb – systems packages are adding to that 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
    - Jessie
  - Ubuntu
    - 16.10
    - 16.04
  - CentOS
    - 7

- Mac OS X
  - El Capitan
- Windows
  - Windows 10 (Ubuntu subsystem) – not tested yet

Using *curl*:

```
curl -sL https://get.frkl.io | sudo bash
```

Using *wget*:

```
wget -O - https://get.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/.freckles/opt/venv_freckles` into which *freckles* and all its requirements are installed (~70mb).

If a `$HOME/.profile` file exists, a line will be added to add `$HOME/.freckles/bin` to the users `$PATH` environment variable. If no such file exists, it's the users responsibility to either add that path, or start *freckles* directly using its path.

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 `pycrypto` 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
```

And freckles also depends on stow (if you want to be able to use that functionality within *freckles*). Either install it via homebrew or ports or whatever. Or from source (check out the stow part of the bootstrap script for an example).

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 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 "$HOME/.freckles/opt/venv_freckles/bin/conda" ]; then export PATH="$HOME/.
↪freckles/opt/venv_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 `$HOME/.freckles/opt`. Symbolic links `freckles` executable as well as some helper applications (`ansible-playbook`, `conda`, etc.) are created in `$HOME/.freckles/bin` and a line is added to `$HOME/.profile` 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

Usage

freckles is using sub-commands, like for example *git*. To see a list of all available sub-commands, use the internal help system:

The internal `help`

```
freckles --help
```

To see the options for each sub-command, use:

```
freckles <subcommand> --help
```

`apply`-ing configurations

`apply` is the main command to be used with *freckles*. It takes one or more configuration files or urls to configuration files, and applies the environment that is described in those onto the machine where *freckles* is running.

```
freckles apply <config_url_or_path_1> [<config_url_or_path_2> <config_url_or_path_3> .
↪..]
```

freckles configuration files are designed to be as simple to create as possible, while maintaining the ability to describe more complex setups by optionally supporting a more complex format. On this page we will only use the simple format, for more details on how to get the most out of it please check out the configuration section, as well as the usage *Examples*.

Installing a few packages

For most simple use-cases, a single configuration file will be sufficient. *freckles* configurations are basically a list of tasks, and each task is of a certain type, and has it's own configuration. *freckles* supports several frequently used task types out of the box, but can be extended to use custom task types if necessary. For a list of existing task types, check here: TBD

The most commonly used task type would probably be the `install` one. Let's assume we only want to use *freckles* to install a few packages using the default package manager. The default package manager for Debian/Ubuntu is `apt`, for RedHat/CentOS its `yum`, and for Mac OS X we use `homebrew`. If we only use one platform, and intend to use the default package manager of this platform, we don't need to specify anything, which makes the configuration file simpler to create, and easier to read.

Here is a config that will install the packages `htop`, `fortunes`, and `zile` on a Debian/Ubuntu system:

```
tasks:
  - install:
      packages:
         - htop
         - fortunes
         - zile
```

In this example, we have one task (of the ‘‘install‘ type), the configuration for the task being a list of package names.

This configuration file (as well as all the following ones) can be found in the main freckles git repository, in the examples folder: https://github.com/makkus/freckles/examples/usage-simple-debian.yml

Because *freckles* is designed to be run as the first command on a fresh install, it can directly download configuration files, without them having to be present locally. It can also use local files though.

Here are a few examples on how to run freckles to apply this configuration, using either a local file, or a remote one:

Local configuration file

Assuming you downloaded the config file into your ‘Downloads’ folder, you can run *freckles* like so:

```
freckles apply ~/Downloads/usage-simple-debian.yml
```

Remote configuration file, full url

Alternatively, we can just provide the full url to the file:

```
freckles apply https://github.com/makkus/freckles/raw/master/examples/usage-simple-
↪debian.yml
```

Remote configuration file, short github url

Because it’s convenient, and easier to remember, *freckles* also supports shortcut urls for files that live on github (other services will be supported in the future):

```
freckles apply gh:makkus/freckles/examples/usage-simple-debian.yml
```

First run output

Either of those commands will do the same, and the output will look something like this:

```
Preparing run #1
Starting run #1

Looks like we need a sudo password for some parts of the pipeline, this might␣
↪interrupt the execution process, depending on how sudo is configured on this␣
↪machine. Please provide your password below (if applicable).

SUDO password:
- task 01/03: apt -> install 'fortunes'      => changed
- task 02/03: apt -> install 'htop'  => changed
```

```
- task 03/03: apt -> install 'zile'  => changed
Run #1 finished: success
```

freckles tries to determine whether a sudo password is required (for example, some package managers need sudo, some other don't, some systems have passwordless sudo, some do not), and it will display the above message if it thinks it is. The password prompt is the underlying *ansible* playbook runs though.

Installing packages using a dotfile repository

This was easy, but most of the time we also have to worry about configurations we want to use on multiple boxes. There are several ways of doing that, each have their advantages and disadvantages. In theory *freckles* can support all of those methods, but at the moment only one is implemented, since that is the one I currently use, and it looks like a lot of other people do too, for example:

- https://alexpearce.me/2016/02/managing-dotfiles-with-stow/
- http://brandon.invergo.net/news/2012-05-26-using-gnu-stow-to-manage-your-dotfiles.html
- http://codyreichert.github.io/blog/2015/07/07/managing-your-dotfiles-with-gnu-stow/
- http://www.garin.io/dotfiles-with-stow

Basically, your dotfiles are all stored in a git repository (here's mine). The folder structure is like:

```
<base-dir>
    |
    |-- app1
    |     |-- .app1rc
    |
    |-- app2
    |     |-- .app2
    |           |-- app2config1
    |           |-- app2config2
    |
   etc
```

This makes for a nice and tidy organisation of all your dotfiles, and they don't get in each others way. In order to get the config files to the location the application expects it to, we use GNU stow. We point `stow` to our base directory, and tell it to symbolically link everything that is in one of the sub-folders of our base directory into the users home directory. `stow` is quite smart and can do that with a few different strategies, but I'll not get into those here. I recommend you look up how `stow` works, it's worth a read.

Since I manage my dotfiles using `git` and `stow` anyway, I figured we can re-use the folder structure we have already to install the packages that belong to our configurations. The only thing we need to do is to name the sub-folders like the package name on the platform we use. As an example, we'll using the emacs-like editor called `zile` which I find quite handy to quickly edit small text files. It uses a configuration file called `.zile`, which needs to be located in the root of the home directory:

```
<base-dir>
    |
    |-- zile
    |     |-- .zile
```

If we cd into the dotfiles base-dir, and run stow with the `zile` argument, this happens:

```
$ cd ~/dotfiles
$ stow zile
$ ls -lah ~/.zile
```

```
lrwxrwxrwx 1 markus markus 19 Apr 20 10:56 /home/markus/.zile -> dotfiles/zile/.zile
$ _
```

I've prepared an example repository, containing an example zile config file here. We'll get *freckles* to checkout this dotfile directory into `$HOME/dotfiles-example`, install all the packages that are named like the sub-folders contained in it (only one in this case, `zile`), and then stow all the config files we need (again, only one). The config to do this looks like:

```
vars:
  dotfiles:
    - base_dir: ~/dotfiles-quickstart
      remote: https://github.com/makkus/dotfiles-example.git
tasks:
  - checkout-dotfiles
  - install:
      use_dotfiles: true
  - stow
```

Applying this config, this is what will happen:

```
$ freckles apply example.yml
Preparing run #1
Starting run #1
- task 01/01: checkout dotfiles 'https://github.com/makkus/dotfiles-example.git -> /
↪home/markus/dotfiles-quickstart' => changed
Run #1 finished: success
Preparing run #2
Starting run #2
- task 01/02: apt -> install 'zile'  => no change
- task 02/02: stow - /home/markus/dotfiles-quickstart/ -> /home/markus 'zile'
↪=> changed
Run #2 finished: success
```

Depending on your environment, it might have also asked for a sudo password again.

Notice how it says `install 'zile' => no change`. This is because we already installed it earlier. Also, notice how the execution is split into two 'runs'. This is because *freckles* needs the up-to-date dotfile repository to exist before it can calculate which applications to install using the folder names within. If we would run all tasks in the same go, no application would be installed because no folder would exist yet (at the time of run preparation).

Also, in the `install` task we have an extra variable `use_dotfiles`. This tells the `install` task to look at the `dotfiles` variable and use the dotfile repo described in it to calculate which applications to install (based on the sub-folder names, as mentioned above), in addition to the `packages` variable (which is empty in this case). This works because *freckles* merges variable dictionaries on top of each other, the closer to the task at hand the later the dict is merged, which means those variables take precedence if there is a conflict. In this example, this means that we give the `install` task 2 variables: `dotfiles` and `use_dotfiles`.

If we check our home directory, we'll see the symbolic link `stow` created:

```
$ ls -lah ~/.zile
lrwxrwxrwx 1 markus markus 30 Apr 20 03:06 /home/markus/.zile -> dotfiles-quickstart/
↪zile/.zile
```

Let's go through the config example and try to understand how it works:

```
vars:
  dotfiles:
```

```
    - base_dir: ~/dotfiles-quickstart
      remote: https://github.com/makkus/dotfiles-example.git
```

This creates a variable called `dotfiles`, which contains a list of dicts as values. The variable(s) created here apply to all `tasks` that are described subsequently. We could also give each task it's own set of variables, like so:

```
tasks:
  - checkout-dotfiles:
      dotfiles:
        - base_dir: ~/dotfiles-quickstart
          remote: https://github.com/makkus/dotfiles-example.git
  - install:
      dotfiles:
        - base_dir: ~/dotfiles-quickstart
          remote: https://github.com/makkus/dotfiles-example.git
      use_dotfiles: true
  - stow:
      dotfiles:
        - base_dir: ~/dotfiles-quickstart
          remote: https://github.com/makkus/dotfiles-example.git
```

In this case this doesn't make sense, since all tasks need the same `dotfiles` variable, and duplicating it would not make any sense.

In general, the tasks we describe are executed in the order they appear in the config file. So, here we checkout the dotfile repo, install all required packages, and finally stow all configurations.

Install packages and execute other tasks

Now, let's merge both ways of installing packages, so we can have both packages that need as well as those that don't need configuration:

```
vars:
  dotfiles:
     - base_dir: ~/dotfiles-quickstart
       remote: https://github.com/makkus/freckles-quickstart.git

tasks:
  - checkout-dotfiles
  - install:
      use_dotfiles: true
      packages:
        - htop
        - fortunes
        - fortunes-off
        - fortunes-mario
  - stow
  - create-folder: ~/.backups/zile
```

In addition to the `checkout-dotfiles`, `install` and `stow` tasks, we introduce a new task type here: `create-folder`. This does exactly what you expect it to do: creates a folder, using a string or list of strings with folder paths. If a folder already exists, it will do nothing.

In this case, we need the folder `$HOME/.backups/zile` because it is configured in the .zile configuration file in our dotfile directory. `zile` itself does not create this folder, and can't create backups if it doesn't exist.

Install packages on different platforms

Depending on your requirements, sometimes you might want to re-create the same environment on different platforms. Say, your development machine is running Mac OS X, but you often use virtual machines running Ubuntu (maybe using Vagrant) as well. One of the problems here is that package names sometimes differ no different platforms. In our last example, we installed the applications `htop`, `fortunes` (including a few debian-specific 'plugins' for it), and `zile`. `htop` and `zile` are usually named the same on most platforms I came across, but `fortunes` is named is called `fortune-mod` on RedHat, and `fortune` on homebrew for Mac OS X.

freckles can handle this, by supporting an optional configuration format for the `install` plugin which deals with more complex contexts. You won't need this too often I'd imagine, but it's simple enough to use to be included in this basic usage guide.

For packages that are named the same, we don't need to do anything in particular, we can leave their config as it is. For the fortune mod we have to tell *freckles* the name of the package(s) on the respective platform:

```
- install:
    use_dotfiles: true
    packages:
      - epel-release:
          pkgs:
            yum:
              - epel-release
      - htop
      - fortune:
          pkgs:
            apt:
              - fortunes
              - fortunes-off
              - fortunes-mario
            yum:
              - fortune-mod
            homebrew:
              - fortune
```

As you can see, *freckles* assumes the package name is the string if the list item under `packages` is a string. If the list item is a dict, it will look for a key called `pkgs` and look up the package manager that is used on the system *freckles* is running on using its key. In the case of a debian-based system, we install 3 packages. Those additional packages don't exist on RedHat or in Homebrew, which is why we don't worry about them. Also, notice how we install the `epel-release` package. This only exists for RedHat-based systems, and is needed to enable some extra repositories without which we wouldn't be able to install some of our specified applications. Since the respective `pkgs` dict does not have entries for `deb` or `homebrew`, this is ignored on those platforms.

For a complete config file that does all of the things we talked about so far, check out: quickstart.yml

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.de>

Contributors

None yet. Why not be the first?

CHAPTER 9

History

0.1.0 (2017-01-05)

- First release on PyPI.

CHAPTER 10

Indices and tables

- genindex
- modindex
- search