PRT455 – Progress Report

Samuel Walladge (s265679)

August 2017

Contents

1 Introduction

This document is a report on the progress made on the dotfiles manager project. It will also present requirements documented from the requirements engineering process.

2 Background

Many software developers and system administrators have a set of configuration files (dotfiles) for their workstations and servers, fine tuned over many years for the perfect experience. Generally, these are stored together in a Git repository or directory of files to back them up, share between computers, and track changes.

There are many existing solutions for managing configuration files stored this way. These range from huge automation products such as Ansible, to small purpose built programs such as Dotbot. However, all of these programs fall short in some area. This project aims to create a dotfiles management program to fulfill the requirements for a good configuration management software by following good requirements engineering practices and an iterative development process.

3 Links

The project is using GitHub as a central place to store the code and manage development. It can be found at `https://github.com/swalladge/dotfiles-manager`.

The Travis CI continuous integration reports are here:
`https://travis-ci.org/swalladge/dotfiles-manager`

The code coverage reports are here:
`https://coveralls.io/github/swalladge/dotfiles-manager`

These services will be discussed further in the report.

4 Requirements

4.1 Functional requirements

Note: IDs of functional requirements correspond to GitHub issues set up for tracking progress.

ID	#1
Title	Work with a structured repository of config files.
Description	The software should support a certain directory/file structure of in the source repository of configuration files.
Rationale	This will allow support for host-specific configuration, modular "packages" of config files, and future expansion, while being simple since it's based on the filesystem.
Dependencies	None

ID	#6
Title	Install a package of configuration files
Description	It shall support installing via symlinks a subset of configuration files from the repository.
Rationale	This is the main purpose of this software, and subsets will allow greater flexibility.
Dependencies	None

ID	#7
Title	Remove a package of configuration files
Description	It shall support removing a previously installed package of configuration files.
Rationale	To allow cleaning up if required, and switching to other methods.
Dependencies	None

ID	#8
Title	Support host-specific sections in a package of configuration files
Description	The software should allow overrides based on the current host.
Rationale	So that differences in configuration per-host can be managed easily.
Dependencies	#6

ID	#9
Title	Support running scripts on events
Description	It should allow for running custom scripts at times, such as after installing a package.
Rationale	This will allow greater flexibility, for things like installing vim plugins after installing a vim package.
Dependencies	#6

ID	#10
Title	Support force installing or removing
Description	It should allow forceably overwriting existing files/links if required.
Rationale	Sometimes existing files are not needed and can be time consuming to manually remove them all.
Dependencies	#6

ID	#11
Title	Support running in a simulation mode
Description	It should be able to run in a simulation or test mode that will not actually make any filesystem changes, only report on what changes would be made.
Rationale	If it is unknown what files would be overwritten or changed, this would be helpful.
Dependencies	#6

ID	#12
Title	Support custom source and target directory
Description	This allows selecting a different directory to install to and from.
Rationale	Greater flexibility and to support non-standard setups.
Dependencies	#6

ID	#13
Title	Support adding/adopting an existing file into repo
Description	It should allow a subcommand to import an existing configuration file into the repository to be tracked.
Rationale	This is to avoid having to manually move/delete/configure when wanting to track a new configuration file.
Dependencies	None

4.2 Non-functional requirements

ID	NFR1
Title	Simplicity
Description	All functionality should be designed with simplicity in mind. Actions should be able to be performed with clear, concise commands.
Rationale	Simplicity is number 1 on the list of what users want, as well as being important for the user interface.
Dependencies	None

ID	NFR2
Title	Reliability
Description	It should be reliable so that nothing unexpected happens, or it leaves the system in an unstable state. This includes lots of testing, regression tests, etc.
Rationale	Leaving the system in a broken state or crashing is bad.
Dependencies	NFR3

ID	NFR3
Title	Tested
Description	There should be an automated suite of tests.
Rationale	This will help with regression testing, and help confirm that everything works.
Dependencies	None

ID	NFR4
Title	Minimal dependencies
Description	It should require a minimum of dependencies to install.
Rationale	Users don't want to have to install extra dependencies just to install their configuration files.
Dependencies	None

ID	NFR5
Title	Maintainability
Description	The software codebase should be easy to maintain and manage.
Rationale	Easier to maintain means that fixing bugs and adding features in the future will be simpler.
Dependencies	None

5 Development

5.1 Configuration management

A version control system, Git, is being used to track changes to the software. For a neat user interface and public hosting, it is also published on GitHub (link in previous Links section). At the time of writing, there are 29 commits in the repository. For reference, my GitHub username is `swalladge`.

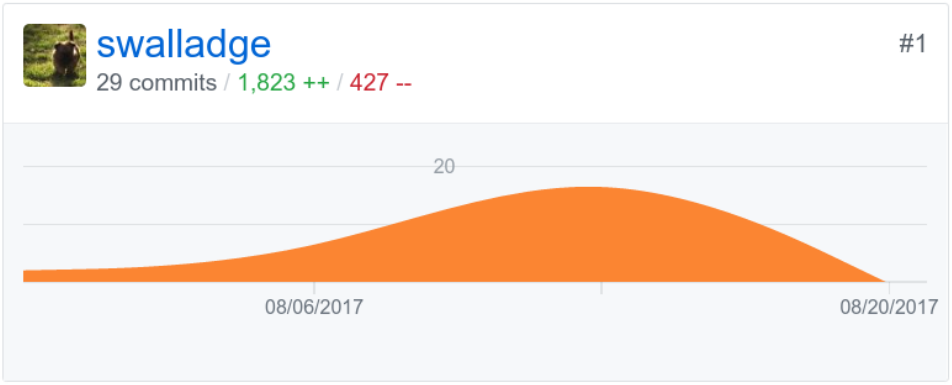

Figure 1: GitHub commits info graph

To ensure builds are reproducible, dependency versions are specified in a configuration

file for `cargo`, the Rust programming language build tool and package manager.

5.2 Technology selection

To begin coding on the project, a target programming language had to be selected. After careful consideration, the Rust language was chosen. There were several reasons for this:

- It is a low level language, providing high performance. (essential for a small command line tool)
- Zero cost abstractions providing features usually only found in high level languages. (good for fast development)
- Object oriented programming and functional programming support. (enables a clean, modular, testable code base)
- Good support for unit testing.
- Good community support to help solve any blocking problems.

It is being developed on a Linux computer, which will be the target operating system for now. Cross platform support is planned, but not currently a priority due to time and resource constraints.

5.3 Testing

Rust has excellent built in support for running unit tests. Tests are currently in place for must of the logic part of the code. A code coverage tool, kcov, and online reporting service, Coveralls, are being used to check how much of the code is being covered by tests.

There are plans to develop integration testing soon to verify correctness regarding file operations. This software's main task is to move, create, delete, link, and otherwise manipulate files. This must be well tested to avoid bugs that cause loss of important data!

5.4 Development style

Once the basic requirements were set, work began on the code to produce a minimal working proof of concept. Now that that is done, the development is following an agile style, including test driven development where possible.

A continuous integration service, Travis CI is being used to run the tests, ensure it builds correctly, and generate the code coverage reports. This is run on every commit pushed to the GitHub repository.

6 Design

After the initial requirements were proposed and presented in the project proposal, I began further requirements engineering to find out more about the target audience and what requirements would actually be required. I created a survey asking questions about what software solutions people are currently using, and looking for good and bad experiences they had. A full analysis of the results was written up and published at `https://swalladge.id.au/posts/2017/08/07/dotfiles-config-survey.html`. This proved very helpful and provided a platform for validating the proposed requirements and discovering new requirements.

A quick summary of the results shows that:

- Simplicity is the most desirable.
- After that, the favourite features were usability, reliability, flexibility, and speed. (All non-functional requirements!)
- Many users developed their own management scripts instead of using an off the shelf solution, mainly (it seemed) because it was simpler to set up.
- Linux was by far the most popular operating system, and thus is the main target OS for this project.
- Many users manage their configuration across multiple computers, showing that there is a need for per-host configuration management features.

- Users didn't like having to install dependencies like Python just to bootstrap their dotfiles. (One of the reasons for using Rust, since it can compile to statically linked binaries for minimal dependencies.)
- As users' setups varied a lot, there were also many suggestions for different features, such as auto syncing, or encryption support.

After this analysis was completed, more work was done on requirements and design to come up with:

- A manageable set of requirements to include in scope.
- An api for the command line program.
- Structure of the configuration files repository to support.
- Organization of code.

6.1 Interface design

The interface is designed as a command line program. This allows quickly typing a command to control it.

A command line interface was chosen mainly because the target user group is generally familiar with the command line, seeing as most configuration that users store in repositories like this are for command line programs. Also, many use this sort of software on servers, where the only available interface is a shell.

The general design of the api follows common command line apis. For example:

```
$ dotfiles-manager -d dotfiles install vim
$ dotfiles-manager --host desktop1 -f install zsh vim
```

6.2 Code design/implementation

The code is still under heavy development, but what has been developed so far includes:

- Commandline argument parsing with the clap-rs library
- Processing parsed arguments into a data structure for use in the program
- Tests for verifying correctness of argument processing
- Continuous integration scripts implemented (for use with Travis CI)
- Code coverage script for local generating reports and for Coveralls
- Initial proof of concept code for symlinking (installing) a package (directory) of configuration files.

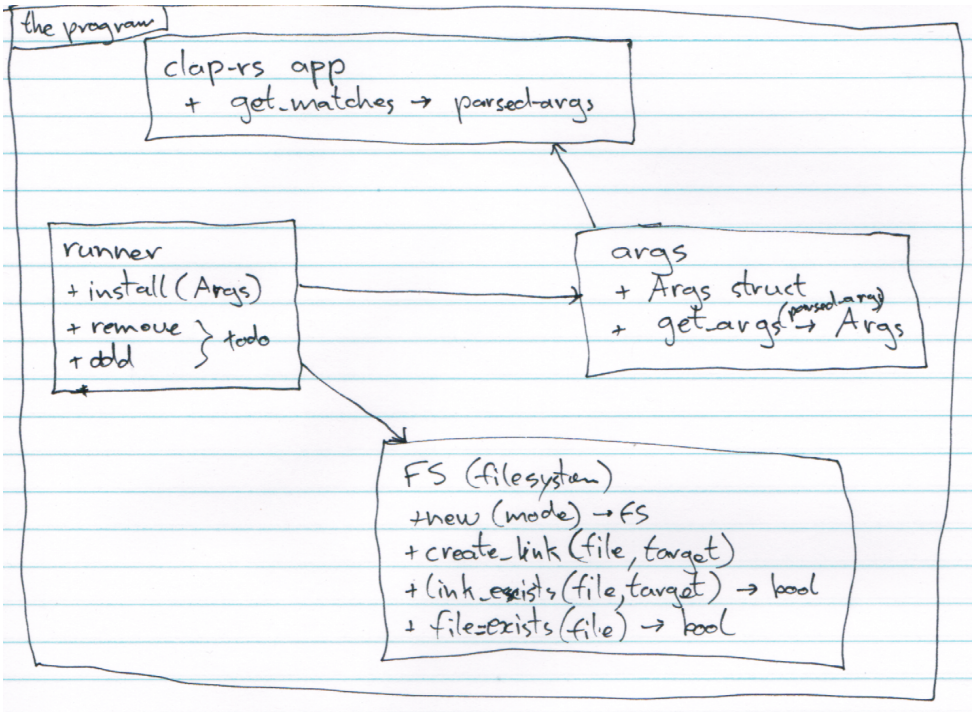

Figure 2: Module diagram of current app code.

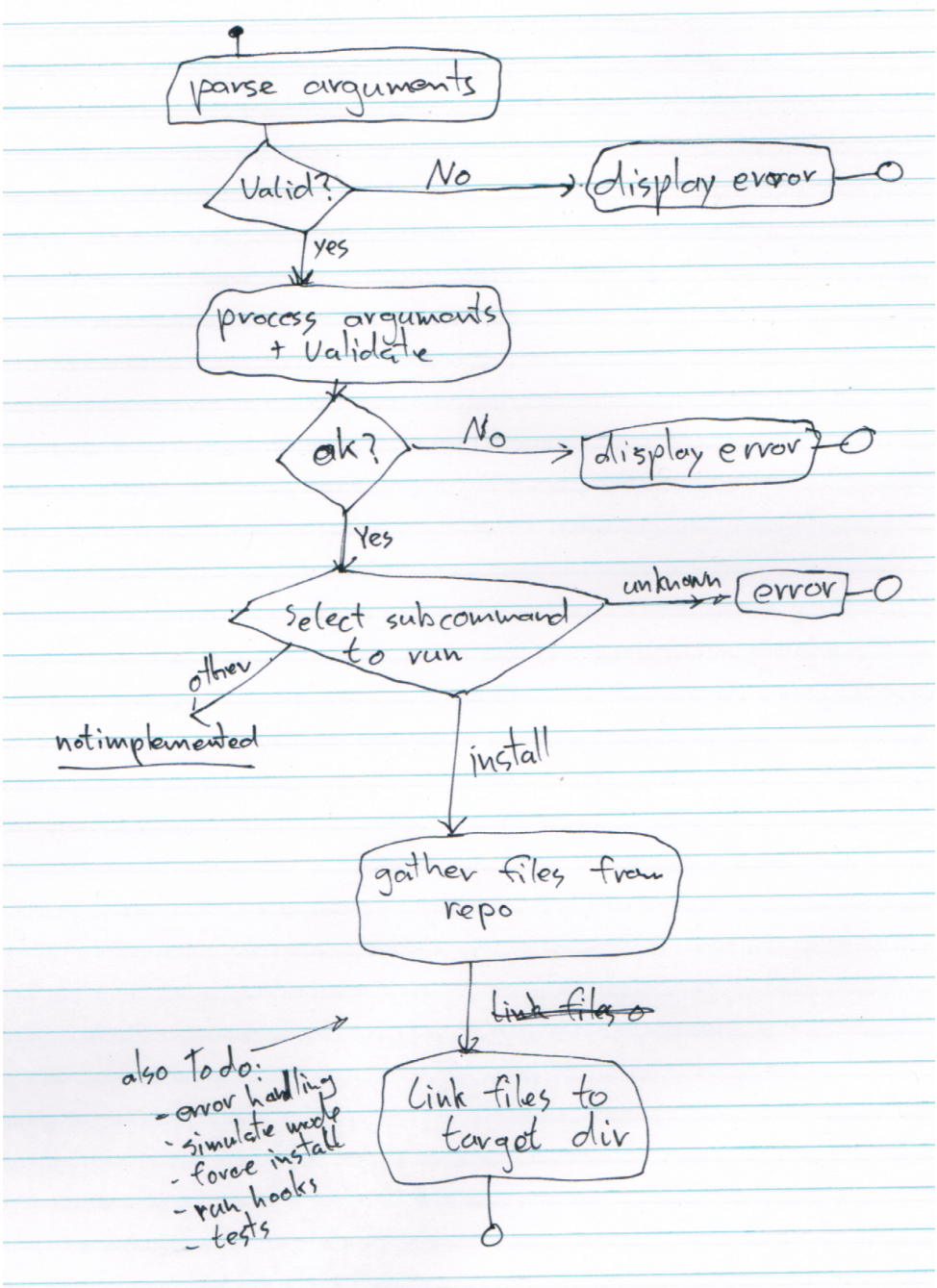

Figure 3: Flowchart of app operations.

7 Plan

The plan for future work on this at this time is straight forward: work on implementing the rest of the requirements, beginning with stabilizing the command line api and a working installation process for configuration files. A TDD process is expected to be followed to

verify everything is working correctly during development.

Also high on the list is a system for testing file operations for it. Initial research has suggested that a custom solution may have to be developed, perhaps using a scripting language and Docker for a safe environment to test it under.