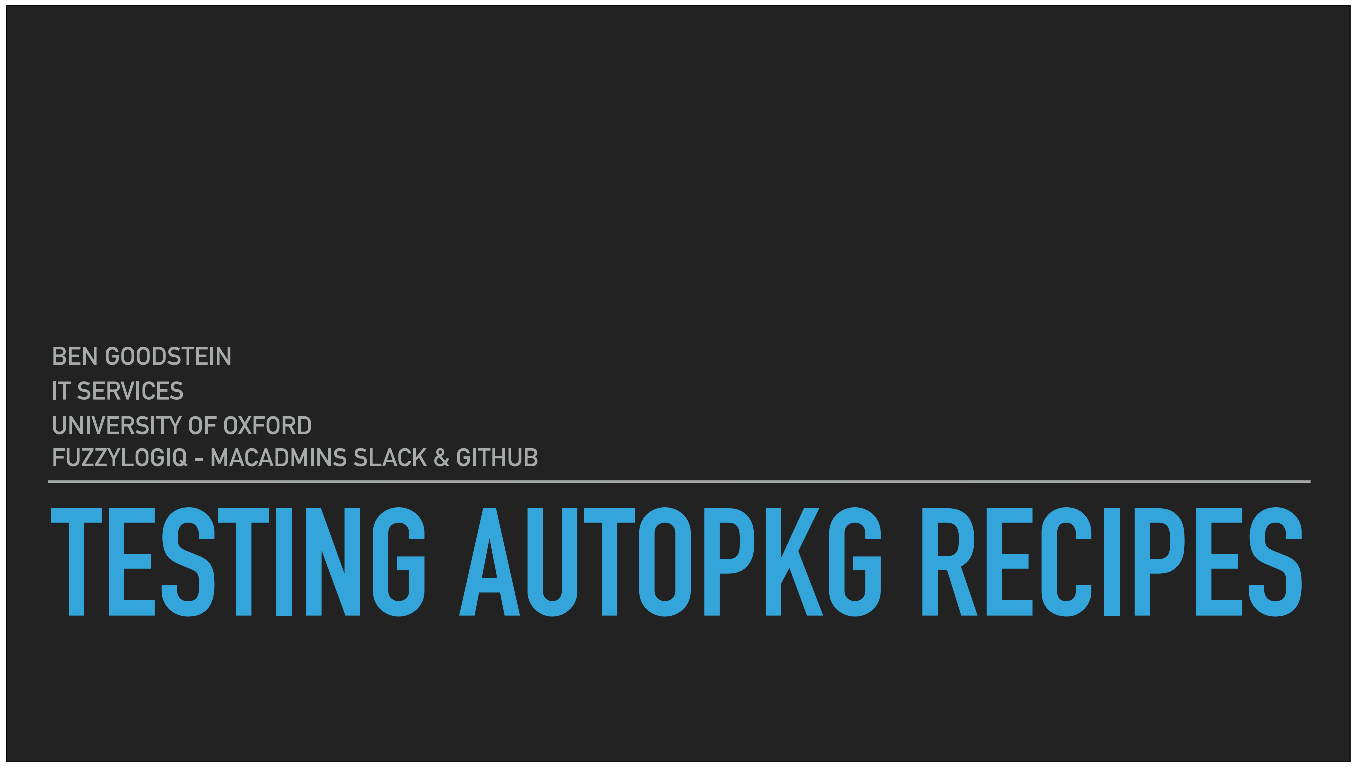
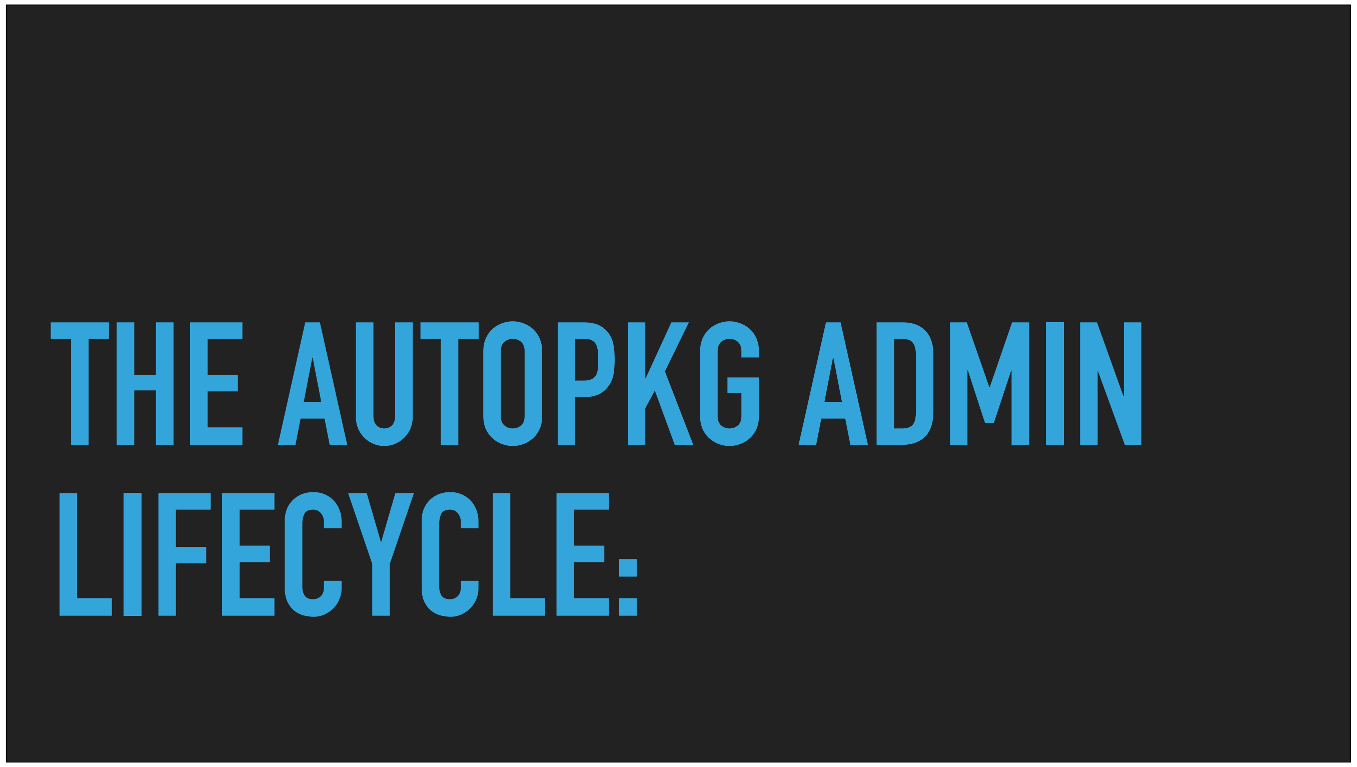


BEN GOODSTEIN
IT SERVICES
UNIVERSITY OF OXFORD
FUZZYLOGIQ - MACADMINS SLACK & GITHUB

TESTING AUTOPKG RECIPES

- Introduce self - lived in London 23 years! 20 min walk to work. :P
- I'm going to assume you all know what Autopkg is and what it's for and I hope most of you are using it (or this is going to be a bit boring!)
- If you're not - good luck!
- I'm also going to assume a little bit of Python knowledge - don't worry though, nothing hardcore, and I'll be explaining any code
- I'm going to talk today about how we are testing submissions to our AutoPkg recipe repos using Travis CI. It may not be hugely relevant or necessary to many of you, but I hope along the way you might find something that is useful or informative!



THE AUTOPKG ADMIN LIFECYCLE:

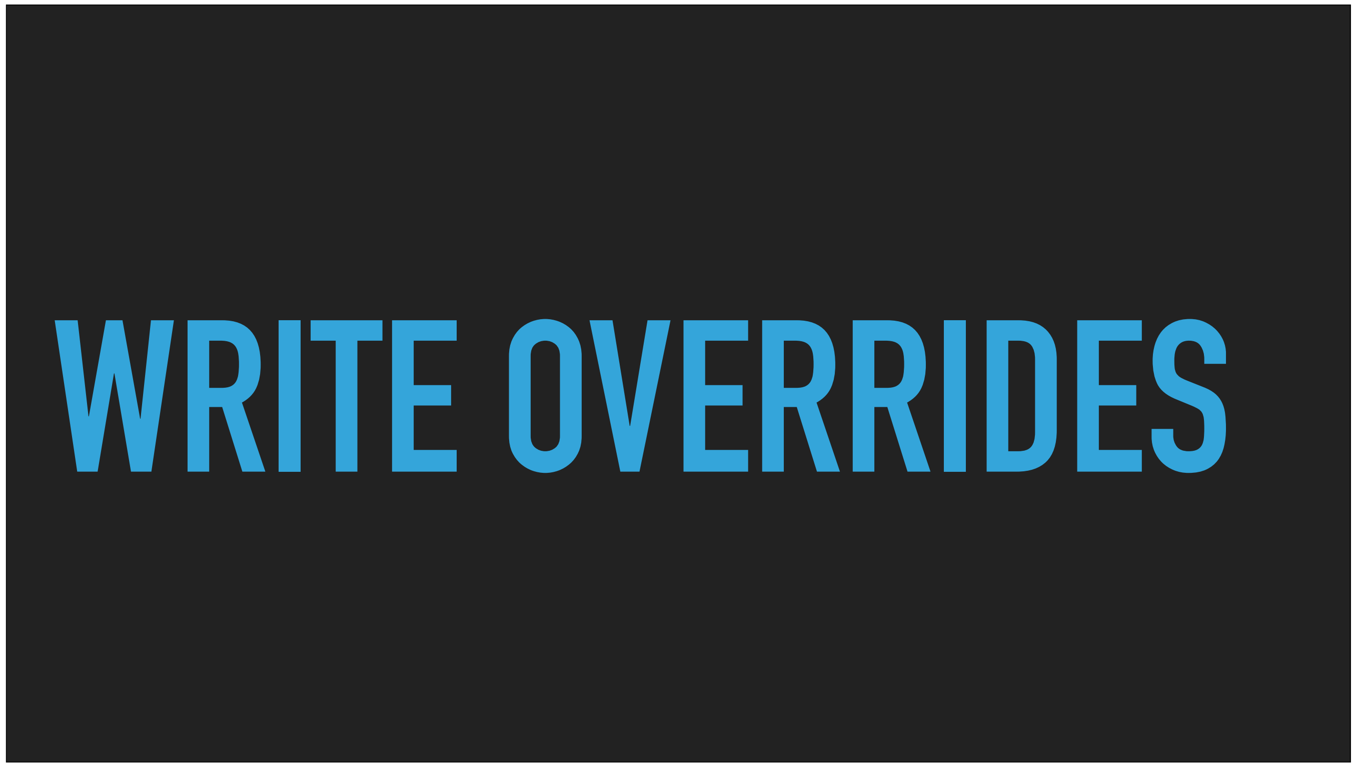
The lifecycle of an AutoPkg admin is pretty much the following

DOWNLOAD AUTOPKG

You download AutoPkg

ADD RECIPES

Then you add some recipes using repos provided by the community



WRITE OVERRIDES

Then perhaps you override them for your environment

WRITE RECIPES

Then maybe you find you have some software to package that there is no community recipe for.
So you write your own recipes.

RETIRE...

And then, having automated all of your software packaging you can retire...

RETIRE...NOT.

Not.

You can then end up with a wealth of plists that you have responsibility for, and it be get quite a lot to keep on top of.

OXFORD IS FEDERATED

- This is particularly a problem for us at Oxford, as we are a largely federated institution.
- As a result of moving to autopkg-powered munki for software delivery we have a pretty large recipe repo
- While we in central IT Services provide a Managed Mac Service called 'Orchard', the colleges, departments etc have their own IT support staff and we encourage them to add to our autopkg repo and thus our munki repo.
- e.g. Computer Science a new client adding Software Development titles

COLLABORATION IS KEY

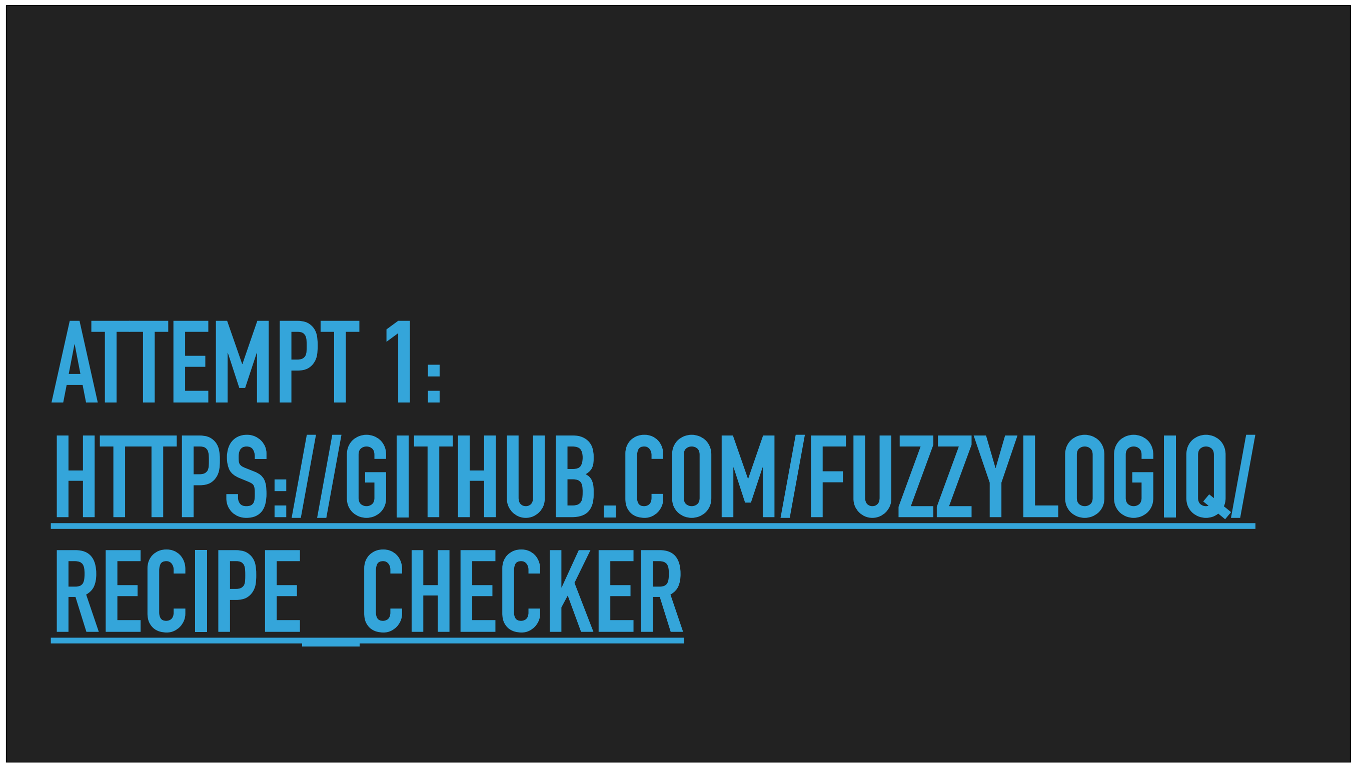
- Since AutoPkg is built around git repositories it allows us to easily accept pull requests from others, or ourselves.
- This lets us maintain one central Oxford repo that everyone can work on. Most autopkg admins might just have one or two people contributing recipes to their repo - we could have 30+

CONSISTENCY IS IMPORTANT

- But then you run into problems enforcing consistency.
- We have guidelines at Oxford which recipe authors know about but how do we make sure they are followed?
  - e.g. if the recipe is going to import into Munki, the pkginfo should have a category, a display name, a developer etc so that the Software Centre looks good.
- We add Attribution dictionary with Copyright/Author info
- We could manually/visually check everything but this is time consuming

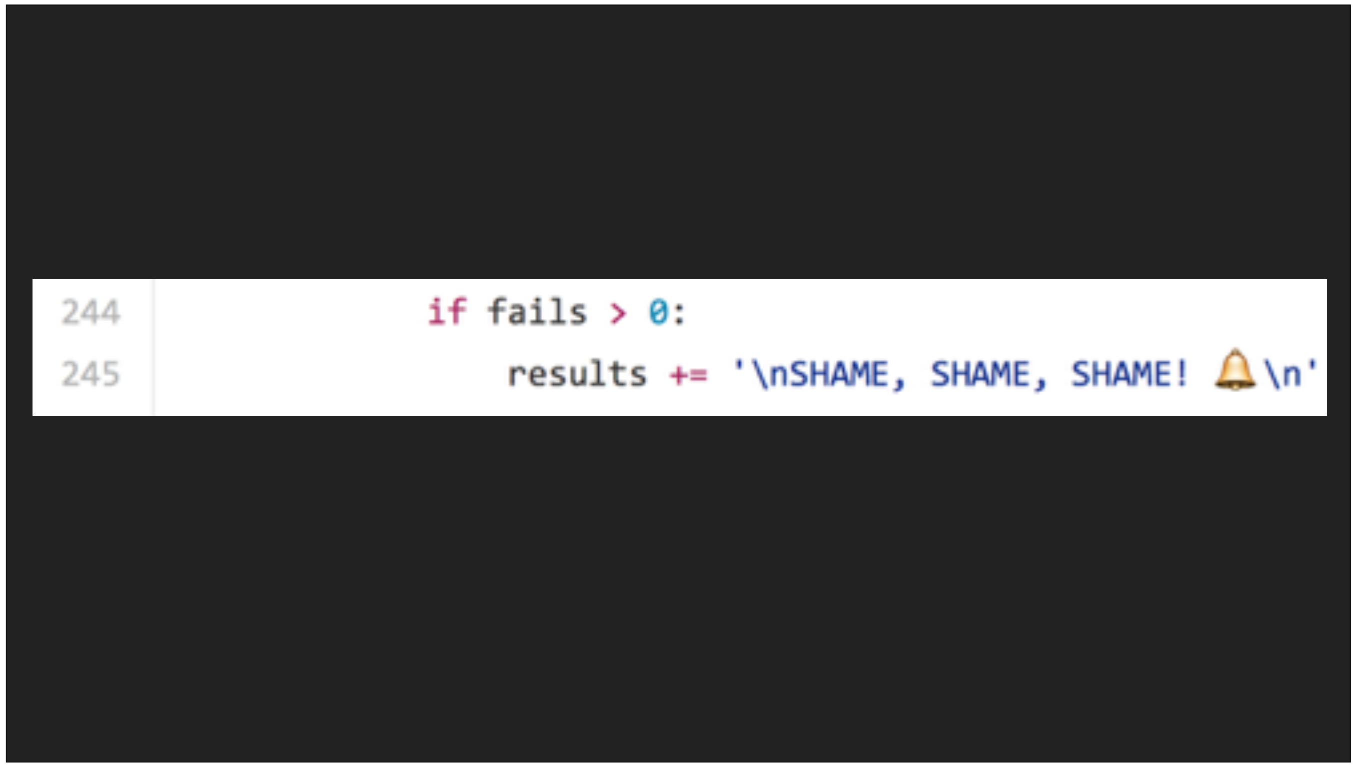
CATCH ALL THE ERRORS

We want to catch errors before they arise on our build host (and we inevitably end up having to fix them)



ATTEMPT 1: HTTPS://GITHUB.COM/FUZZYLOGIQ/RECIPE_CHECKER

- Spurred on by a few examples I heard about in Slack, I started to try and write tests for our recipes.
- You can see some of my early attempts here
- In the beginning we thought to just run them locally on recipes and possibly email the author
- I tried various things...from lots of tests in a class, to a class that could take a proprietary plist of tests and run it on a recipe.

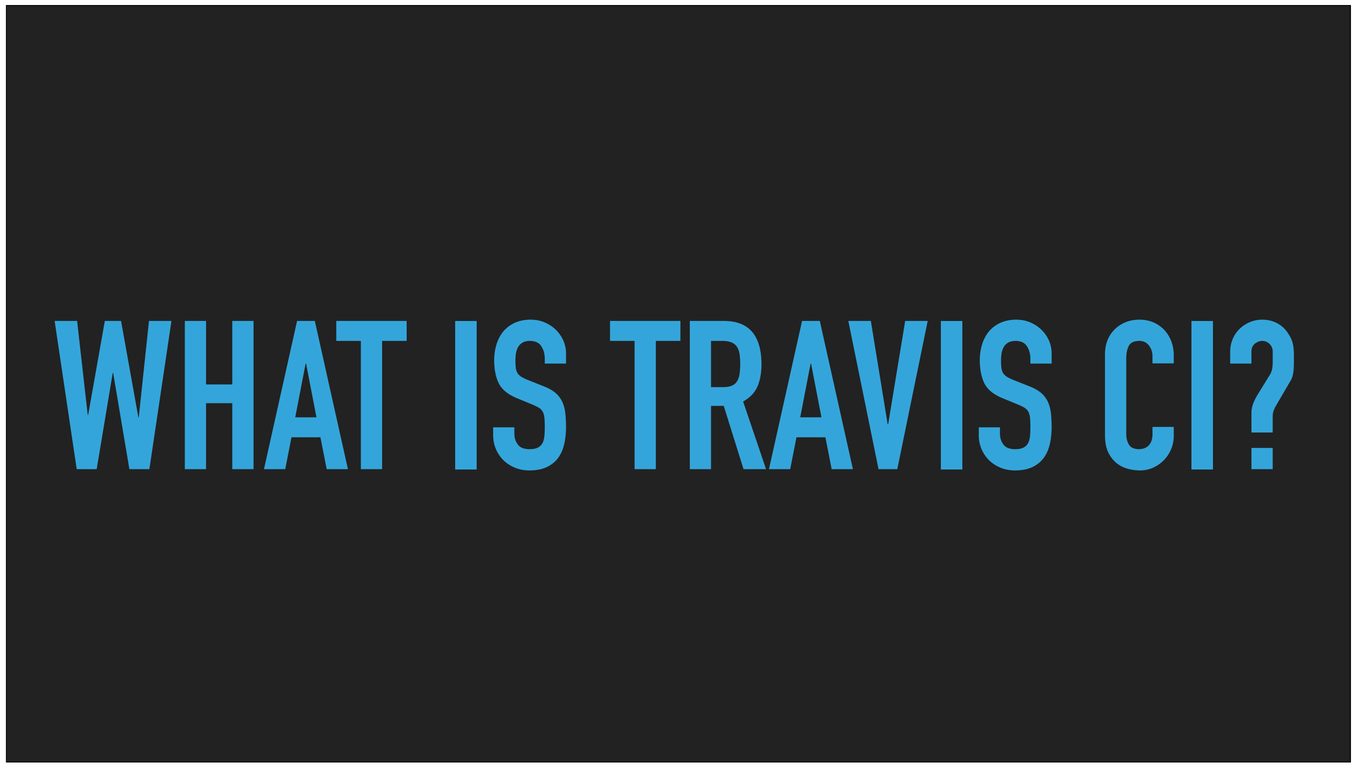


```
        if fails > 0:
            results += '\nSHAME, SHAME, SHAME! 🔔\n'
```

It contained this Gratuitous Use of Emoji that I thought was very amusing but my boss, not so much.

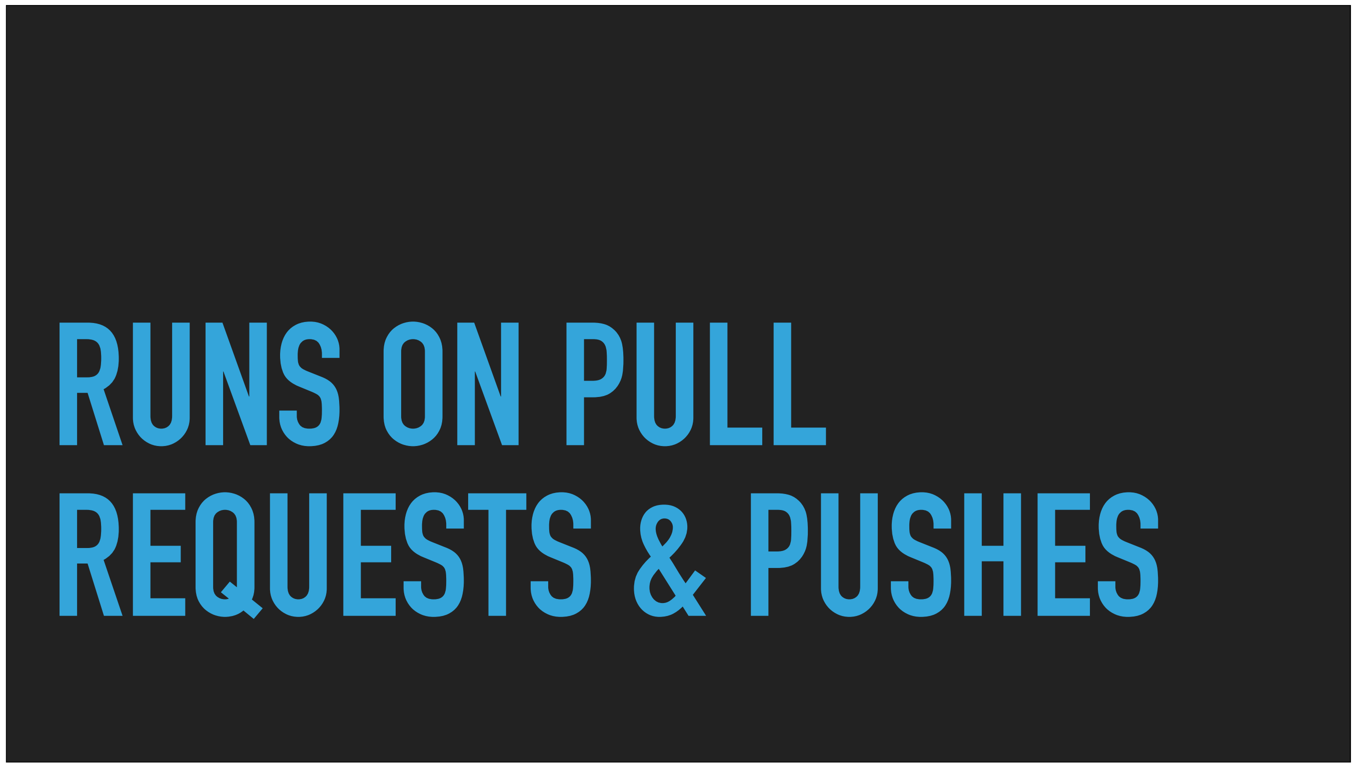
HTTPS://TRAVIS-CI.ORG/ HJUUTILAINEN/RECIPE-CI-TESTING

- Then I happened upon what Hannes Juutilainen (author of Munki Admin) was doing with Travis CI.
- He was actually running autopkg recipes in Travis CI each time they were checked in, in order to do virus checking on them.
- But this got me thinking about Travis CI and what it could do.



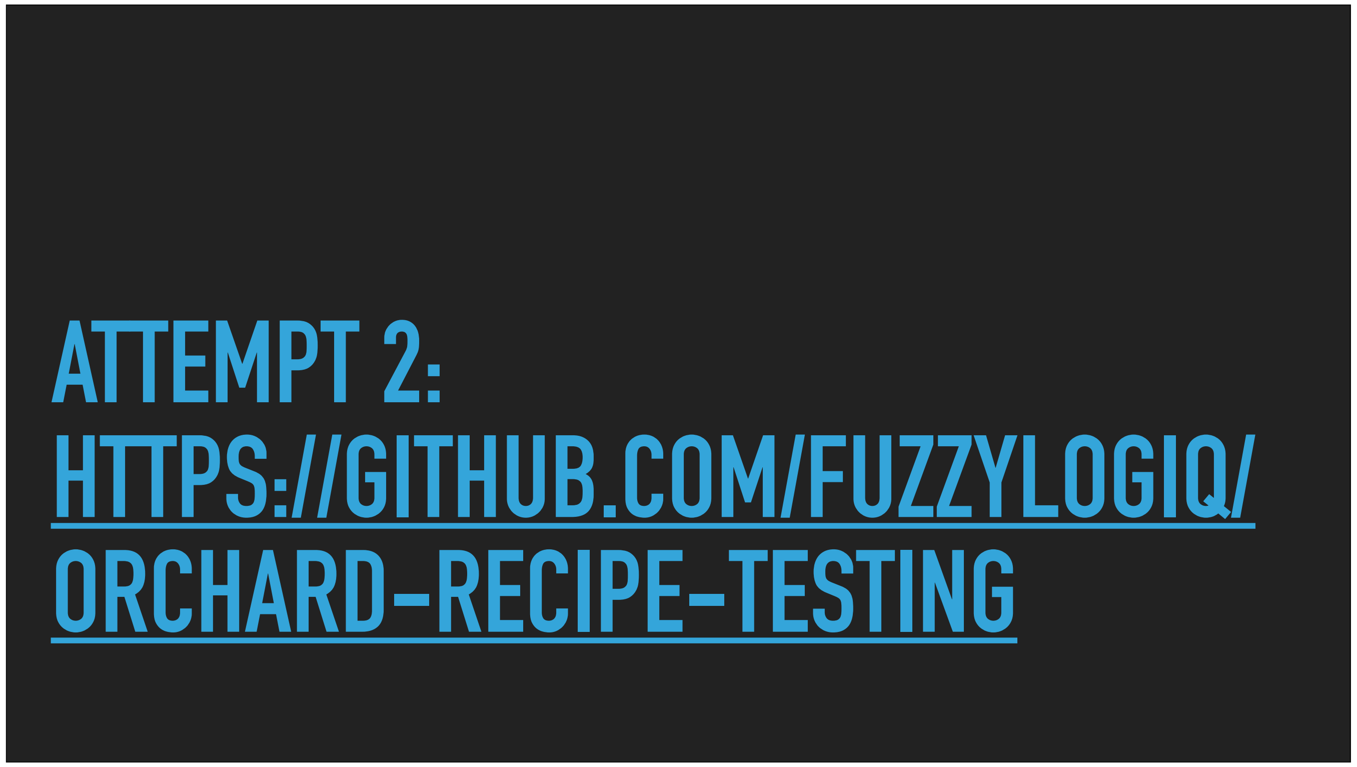
WHAT IS TRAVIS CI?

- CI stands for continuous integration. It's really intended for checking that an application's functionality is not broken by new code.
- But the key thing for us is it didn't have to test an application, it could test anything.
- It's free for open source Github projects (those that are 'public')
- At its most basic it spins up a Docker container each time a change is made or proposed to the repo, and runs arbitrary tests of your choosing on the code.
- Has a nice shiny web GUI, REST API, all the stuff you want.
- Can run Linux or OS X containers (OS X is actually a server farm full of aging XServes)



RUNS ON PULL REQUESTS & PUSHES

- Since Travis can be set to run on both pushes to the repo and pull requests, it means that if either our main team pushes to the repo or others make pull requests, the tests can be run to ensure consistency of the whole recipe repo.
- You can see directly in the Github web interface whether a pull request passed the tests or not.



ATTEMPT 2: HTTPS://GITHUB.COM/FUZZYLOGIQ/ORCHARD-RECIPE-TESTING

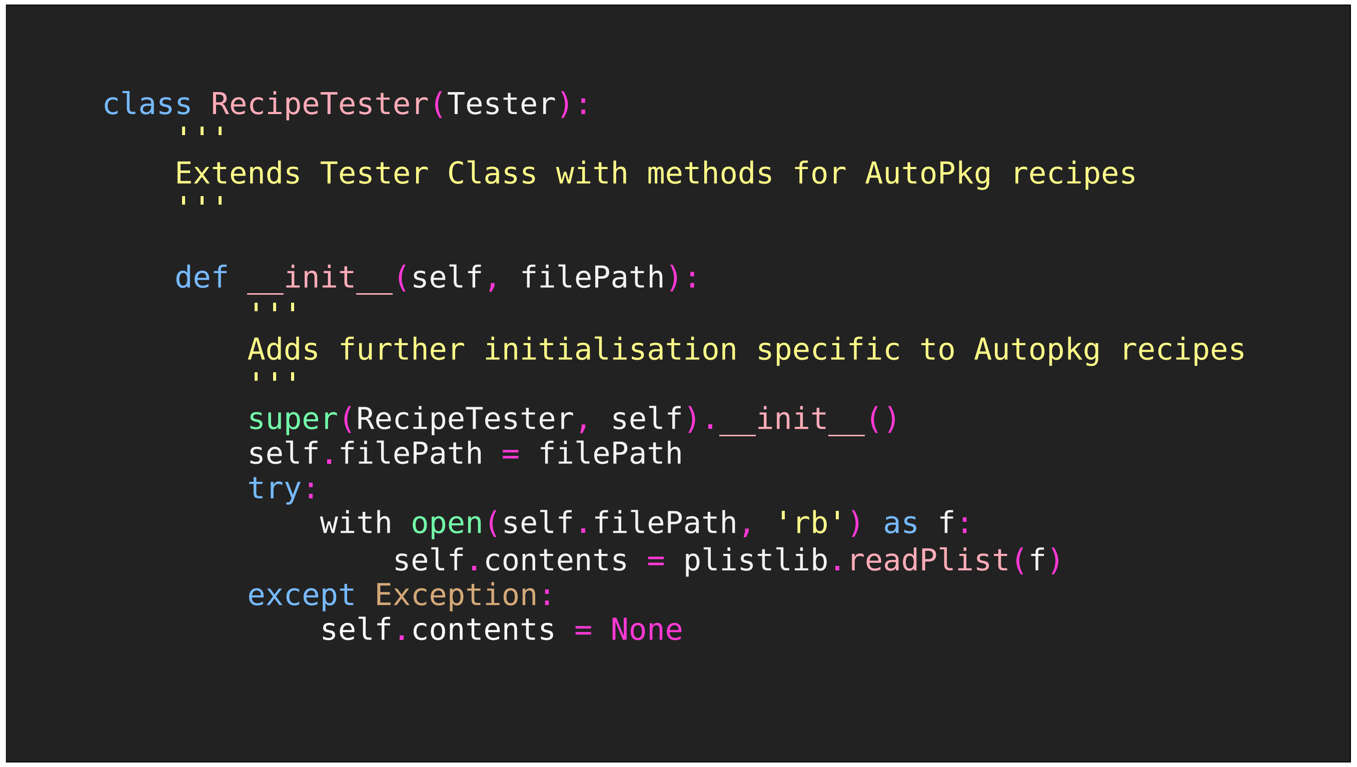
- So we come to the current iteration of my tests, which have been written with Travis in mind.
- They are based on the python unittest framework, and have very similar output, but can take recipes as input instead of python code.
- It's a python recipe testing class that can be subclassed to add different types of assertions, and run these assertions on various bits of the autopkg recipe.
- In the .travis.yml of a recipe repo, you can get travis to clone the tester code and run it on all the recipes in the repo.
- This will be moving to the OX-IT Github once in production!

STOP! DEMO TIME...

1. I've set up a demonstration repo with a couple of AutoPkg recipes in it. Show the .travis.yml
2. I'm a contributor who has forked the repo and done a pull request but it's failed! Let's see why...(show travis)
3. Revert the change, commit and go back to the pull request
4. Show the PR waiting for the build
5. Go through the code

```python
class Tester(object):
...
def run_tests(self):
    ...
    tests = [f for f in dir(self) if f.startswith('test')]
        ...
        for test in tests:
            try:
                result, msg = getattr(self, test)()
            except Exception as e:
                tb = traceback.format_exc()
                result, msg = 'error', str(e) + '\n%s' % tb
```

- The abstract-ish base Tester class performs the tests, evaluates their assertions and prints out the results, exiting with non-zero if there's any fails or errors.
- It runs every method in an instance of the class that begins with the word "test"
- These tests assert something simple like the something is True, or something contains something else



```
class RecipeTester(Tester):
    '''
    Extends Tester Class with methods for AutoPkg recipes
    '''

    def __init__(self, filePath):
        '''
        Adds further initialisation specific to Autopkg recipes
        '''
        super(RecipeTester, self).__init__()
        self.filePath = filePath
        try:
            with open(self.filePath, 'rb') as f:
                self.contents = plistlib.readPlist(f)
        except Exception:
            self.contents = None
```

- The RecipeTester class then adds a bit of extra functionality specifically for the testing of AutoPkg recipes
- It adds the concept of a filePath, which lets you run tests on the name of the recipe itself, and tries to load the plist at that path into 'self.contents'
- And a few methods which are specific to the testing of recipes

```python
class OrchardMunkiRecipeTester(RecipeTester):

    def test_recipe_has_munki_extension(self):
        return self.assertTrue(self.getRecipeType() == '.munki')

    def test_input_pkginfo_category_not_blank(self):
        return self.assertDictContains(self.contents,
                                       ['Input', 'pkginfo', 'category'],
                                       expectedValue=self.NOTBLANK)


        ...
```

- Then when you subclass the RecipeTester class you can write the actual tests themselves
- Talk through the examples here

MEANWHILE, BACK ON DEMO GOD OLYMPUS...

- Go back to github - show that the build has now passed
- Show the Travis output no longer erroring

NOW I CAN RETIRE

(JUST KIDDING, BOSS!)

REFERENCES

- https://github.com/ox-it/orchard-recipe-testing (Future home of recipe testing stuff when in production)
- https://github.com/autopkg/orchard-recipes (Our public-facing recipe repo)
- https://github.com/autopkg/jss-recipes/blob/master/validate_recipes.py (Elliott Jordan's JSS recipe validation script that got me thinking)
- https://docs.python.org/2/library/unittest.html (Python unittest framework)
- https://github.com/mjung/publications (Marko Jung's talks for some background on Oxford University's peculiarities)