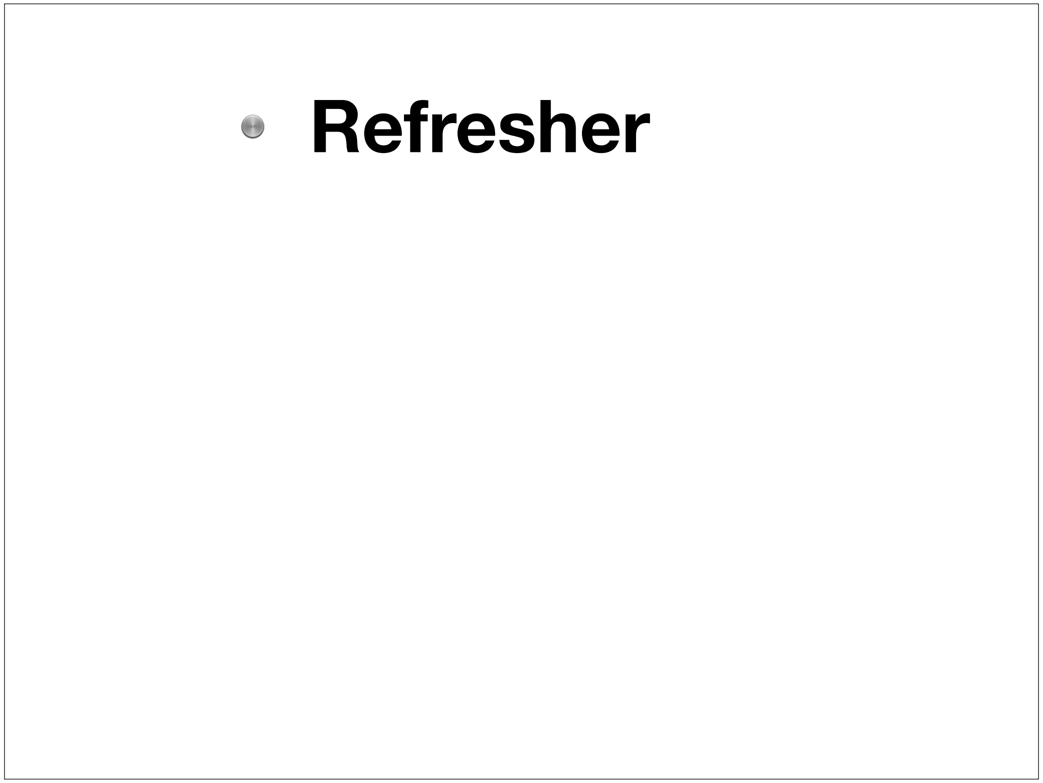
Autopkg: Beyond the Basics

Allister Banks

@sacrilicious

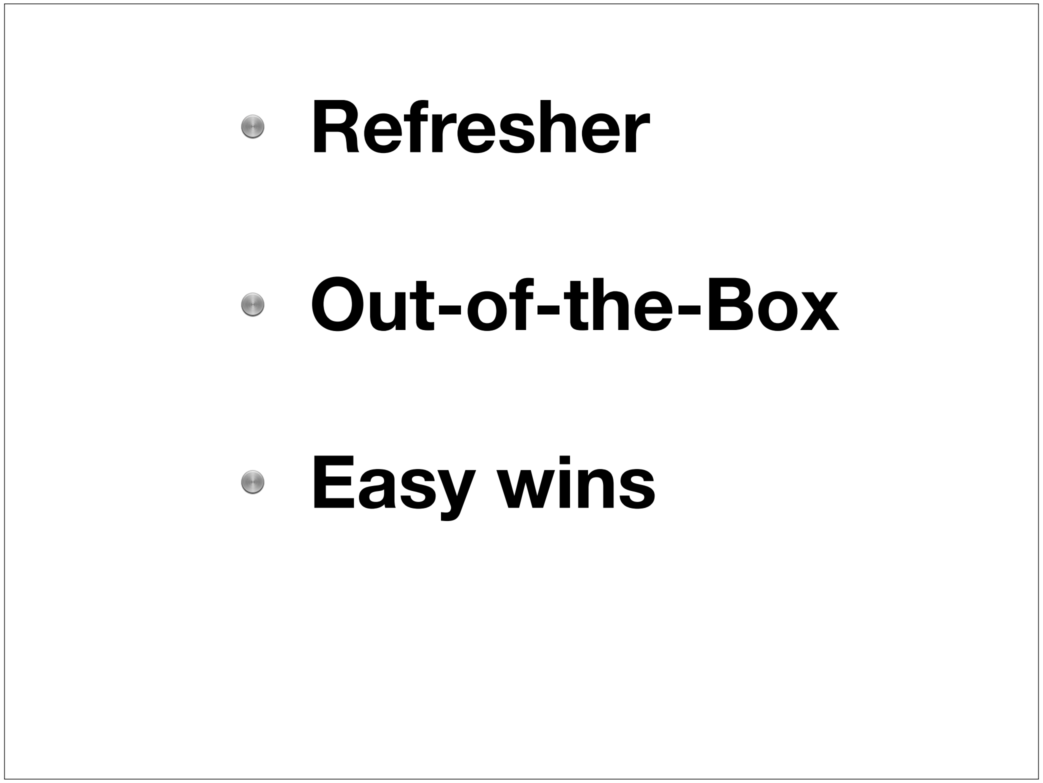


- **Refresher**

Getting down to business, I didn't want to assume any particular level of familiarity with the topic, so my talk aspires to bring everyone who needs a refresher on autopkg into the fold and up to speed, but then I really want to blow the doors off. Besides an abbreviated intro to autopkg, here's the other sections I've broken this talk down into:

- **Refresher**
- **Out-of-the-Box**

- Interesting Out-of-the box benefits that I believe a lot of us can consider taking advantage of



- Refresher
- Out-of-the-Box
- Easy wins

- Ways in which even the most command-line averse or code-shy of us can do minor things to the text of an autopkg recipe in order to pitch in and help out, along with some background on bending vendors that you want to have integrate with autopkg to your will, and an onramp or easy way to jumpstart recipe creation

- **Refresher**
- **Out-of-the-Box**
- **Easy wins**
- **Don'ts**

- And then finally, a cautionary tale of what not to wear:

or, don't be like me, extend autopkg functionality the right way and for the right reasons

Per&
Tim&
Greg.

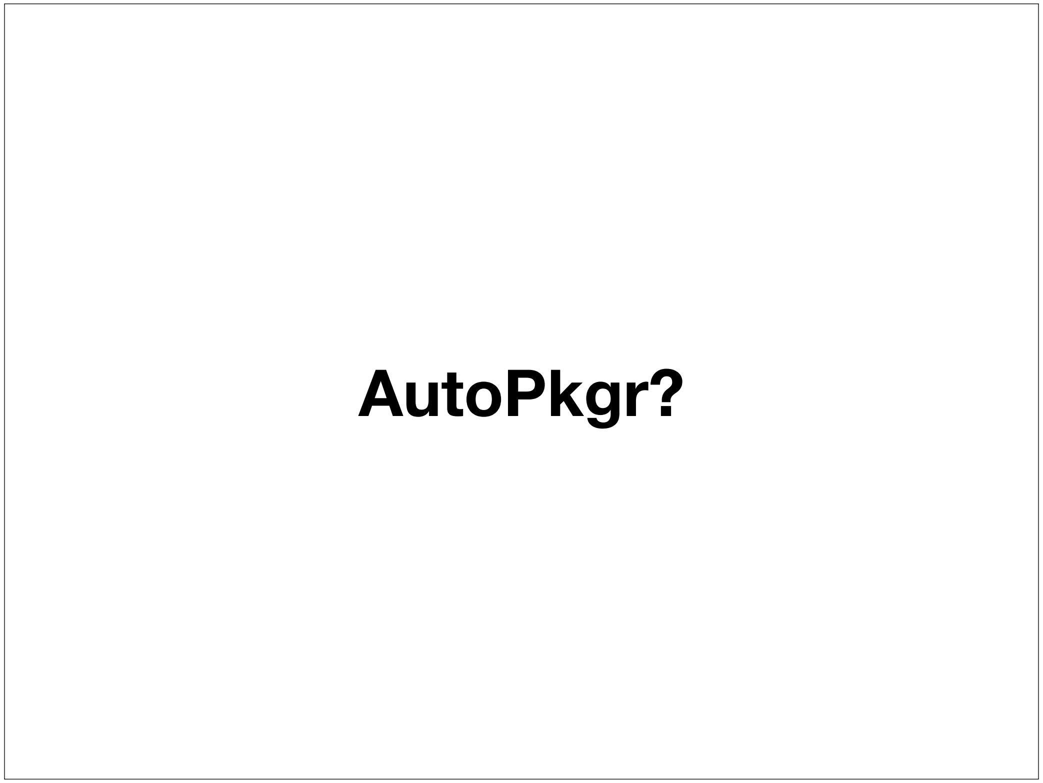
Here's my one slide intro to the history of the project: Per Oloffson, Tim Sutton and Greg Neagle, combined forces, from three different time zones, 3 different countries and over the course of the past 4 years, to build up autopkg into a combination of:

1. check for update

intelligent update feed scraping and

2. make (more) deployable

tasks which leverage their combined expertise and tooling around software deployment, or, to put it another way, patch management. Most of this did grow out of assisting some of the guesswork or grunt work that goes into the support and development of the munki project, but it's tried to stay agnostic, as a more all-purpose tool. Per, for example, happens to primarily use AbsoluteManage at his institution.



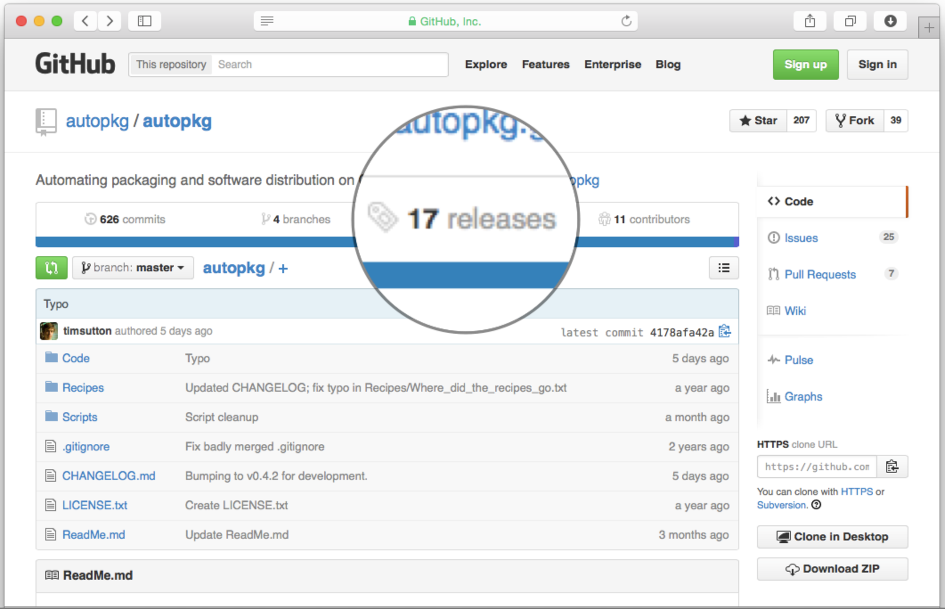
AutoPkgr?

We'll be overlooking the Autopkgr project from the Lindy Group up in San Fran when talking about getting started, which admittedly would be friendlier and certainly faster for many, but over the course of this presentation I'll be taking advantage of things that are closer to the bare metal, and I want autopkg's shiny robotic core of text files to be visible.

http://www.lindegroup.com/autopkgr

Elliot Jordan gave a great presentation on AutoPkgr at the JNUC, jamf national user conference last month, which is available to view online now. And they discuss munki being a system AutoPkgr integrates with as well.
However, for us, the bootstrapping steps on a completely fresh system are

releases



Automating packaging and software distribution on

You can clone with HTTPS or Subversion.

Typo

timsutton authored 5 days ago

latest commit 4178afa42a

Code	Typo	5 days ago
Recipes	Updated CHANGELOG; fix typo in Recipes/Where_did_the_recipes_go.txt	a year ago
Scripts	Script cleanup	a month ago
.gitignore	Fix badly merged .gitignore	2 years ago
CHANGELOG.md	Bumping to v0.4.2 for development.	5 days ago
LICENSE.txt	Create LICENSE.txt	a year ago
ReadMe.md	Update ReadMe.md	3 months ago

step one, grab the latest autopkg release and install it

Xcode CLI tools || git pkg +

`defaults write`

step two either install the xcode command line tools, or both install git and run a defaults write command that you can copy-paste verbatim from the autopkg wiki, and then

```
autopkg repo-add recipes
```

three add some repo's of recipes which will help you fetch the most updated releases of common software like firefox

My gauge for a good open source project is if I can get started in less than 15 minutes and about three rough steps, which autopkg clearly satisfies. That gets us ready for the main event,

___, Forrest!

just run it from any old standard user, it does not even require admin privileges, in the same model as how munki runs for end users.

live to serve

The advantage of not requiring a GUI to run is that you can feel all 20-th century and use...
you can have a service user account that runs the process in the background and does the heavy lifting for you to grab these updates, and it's as reliable as... computers. But, one would consider them less forgetful and better suited to the labor of repeatedly checking and notifying us when it fetches our newspaper.
On top of that, you can approach a nearly no-touch setup where the software products you support or want to maintain are imported directly into your software distribution repository and just about immediately ready to be tested before deploying more widely.

feed the monster

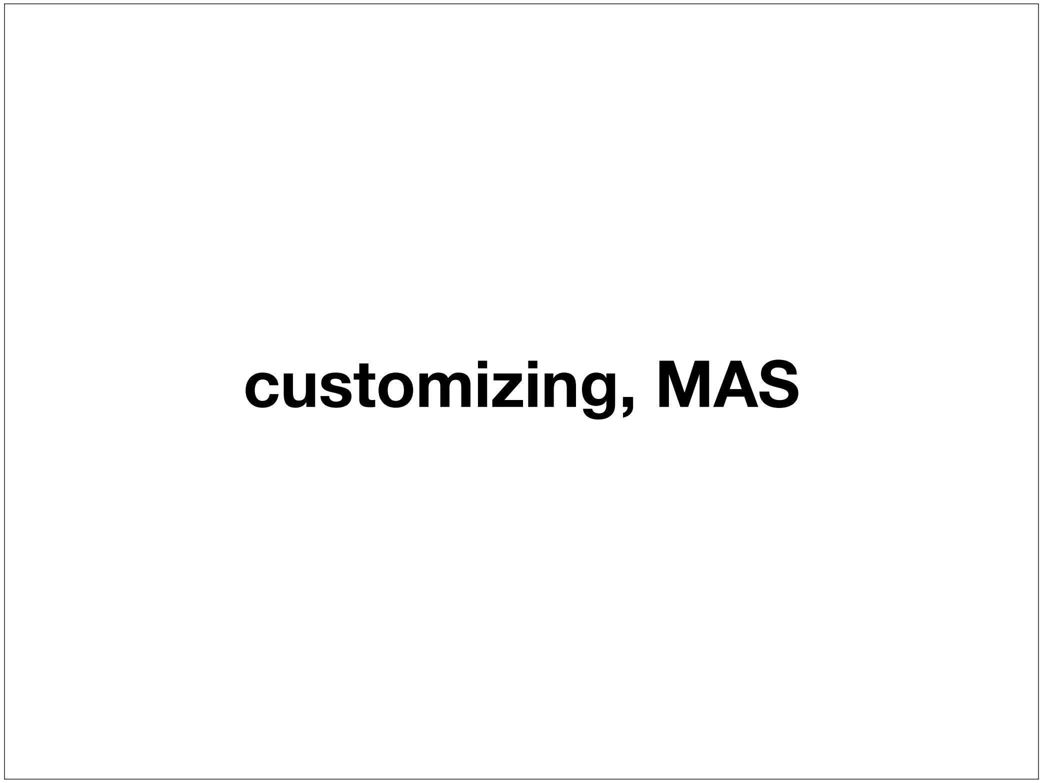
You can use autopkg to import software, with various degrees of smoothness, into the JAMF Casper Suite's JSS, AbsoluteManage, IBM Endpoint Manager (aka Tivoli or BigFix), DeployStudio's raw package directory as part of its repo, just in the filesystem it consults when building workflows, and there's even been talk of SCCM for Macs. Who knows if there's even corners of the universe where FileWave or Radmind are in use and actively that may receiving the output of autopkg in a deployable fashion.

…eau my!

But I'm a longtime Munki proponent, so I apologize if I espouse my opinion of it being the best system to use, even though many of the topics I'm going to discuss are mostly applicable to any software or patch mgmt 'platform':
So recapping that autopkg primer or refresher, it 1. fetches the most recent release, or version, or patch for various software projects by looking at its update feed, then once it pulls it down locally it 'sanitizes' it as necessary for mass consumption,

http://upload.wikimedia.org/wikipedia/commons/7/75/254_TShirt_Cannon.jpg

and then can even load it into your t-shirt cannon to cast a net over your QA or designated testers for them to give you feedback or just the a-ok to distribute it to your entire fleet. Two workflow points that come up as common 'how do I work this thing' questions



customizing, MAS

once we've got it out of the box are: what if I don't like the defaults the recipe creator chose, and while it seems to be discussed more often in confusion rather than when expressing a succinct practical need, folks want to know how to distribute apps that are only available from the MacAppStore.

make-override
(no sudo required)

First, as some of y'all that have tinkered with autopkg's innards may already know, the magic word sudo make me a sandwich in autopkgland is make-override. This presents you with a set of variables in use when a given recipe runs, which you may have seen are often in the Input stanza of a recipe, and you can therefore change, or override what the end result will be.

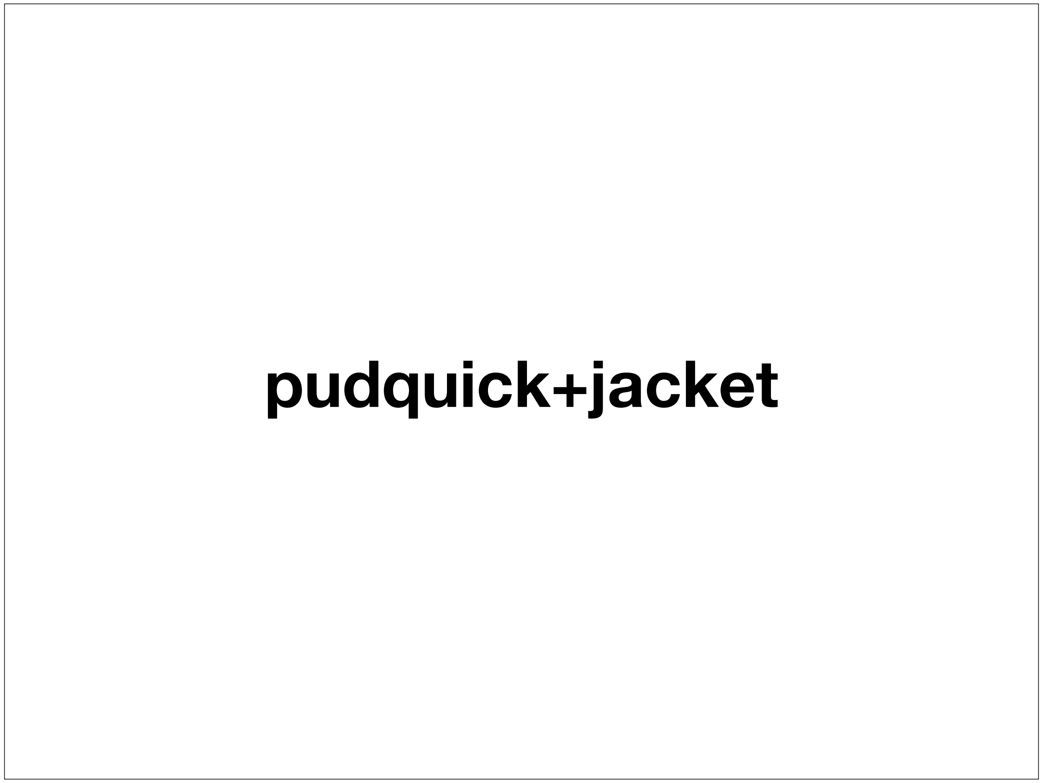
This becomes critical to the other topic, distributing MacAppStore Apps with the help of autopkg. And Greg touched on it briefly during the preso Tim and him did at MacSysadmin…

delivery

If you were not aware, apps from the MAS -are actually shipped to your system as a flat pkg with little other than the app bundle as its payload, but intercepting it isn't where we're going to go for this use case.

As plenty of patch mgmt systems can throw any self-contained or drag&drop app into a dmg or archive to distribute it, your next question SHOULD be, where does autopkg come in to all of this. Well it starts with what we said was the first step of autopkg, which is to check for an update.

If you can properly feed itunes.apple.com the metadata for a given MacAppStore apps, you can compare the one on disk with what the current version is.



pudquick+jacket

The code for performing that check originated with Michael Lynn, aka pudquick on the githubs, who will be speaking later on today, and NickMcSpadden 'weaponized' it by making a recipe for a generic Apple AppStoreApp in his recipe repo.

As you may know you can only interact with Mac App Store updates in one of two ways: either let the store autoupdate, or wait for autopkg to run that check for you, then manually feed the store whichever institutional apple ID you use to pull packages for distribution down with,

at which point you'd get that new DRM wrapped version into a pkg, and as a further option of his recipes, tossed into munki.

manually get down,
automatically wrap up

So to reiterate that work flow, overrides are where you'd customize Nick's generic recipe to tell it the product you'd like imported, and as you manually bring updates down to an admin machine with the autopkg tools on it, you can automatically wrap it up for distribution.

with great power

I hope we can appreciate this AppStoreApp suite of recipes as an option available to us, but as with sudo make me a sandwich or spiderman, with great power comes great responsiblity.

implementation...

In discussing this technique I'm side-stepping two points: one is controversy or general heartburn over interacting with apps that have DRM from the MacAppStore, but that's been discussed on the MacEnterprise list ad nauseum. You would need to be taking responsibility for deploying things in a way apple did not explicitly endorse, so talk to your legal department and recognize that using institutional apple IDs without proper control can have annoying effects.

Like when that one customer who sees an update in the MacAppStore, because you didn't disable the notification, goes ahead and innocently tries to reset your password enough times that you get locked out.

The other is the current situation where certain apps may not allow distribution in this way, based on the receipt validation they perform to ensure they're running on the system they were downloaded from. It's the developers prerogative and right to enforce that as they may, and you do need to take into account and test for any issues that updates cause. Test, test, test. The AppStoreApp recipes may be attractive, though, If VPPv2 is either still unavailable in your country or impractical for you and your only recourse is to distribute an app in this way.

foo.installer.recipe

Possibly of more wide-ranging utility to admins is using another newer feature enabled for autopkg, and that is the concept of an installer recipe. Greg Neagle the bagel wasn't actually interested in me espousing this practice, but I think for sysadmins especially it's kindof nice.
An installer recipe can build on other recipes and, even if you're running it as a standard user,
it can leverage the daemon autopkg uses for certain functionality and therefore get the rights on your behalf to copy apps to your Applications folder,
or otherwise install pkgs onto the local running system.
At this point, I'm totally over looking for download links for apps on the internet anymore, the search option you can pass autpkg will even check a bunch of recipe repos for you.

foo.dmg/pkg…

The current limitation of this installer idea is that it either takes software downloaded as dmgs, or uses pkg recipes as a parent to inherit, but I've had talks with folks about expanding it to support zips.

That's really all I have to say about it, I'm trying to popularize its use

homebrew/cask (caskroom.io)

Homebrew

Cask

“To install, drag this icon…” no more!

Homebrew Cask extends Homebrew and brings its elegance, simplicity, and speed to OS X applications and large binaries alike.

It only takes 1 line in your shell to reach 2007 Casks maintained by 964 contributors.

```
w cask install google-chrome
```

Install with ease

Your software is just one command away from being r
and raring to go.

Forget all about babysitting the install process step by
from website to cleanup.

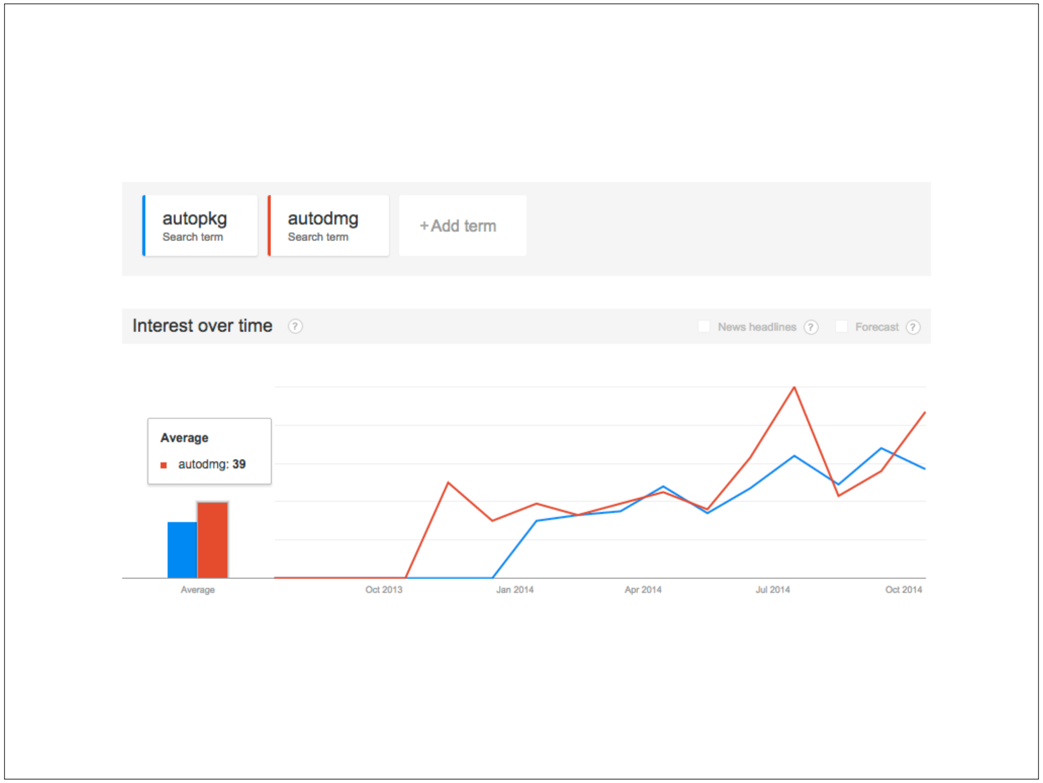
in contrast to tools like homebrew cask, which you may have heard of,

that I find issues with when it comes to reliably updating, I'm sure it's tricky for them as they have a high volume of included software and a flood of activity to keep track of.

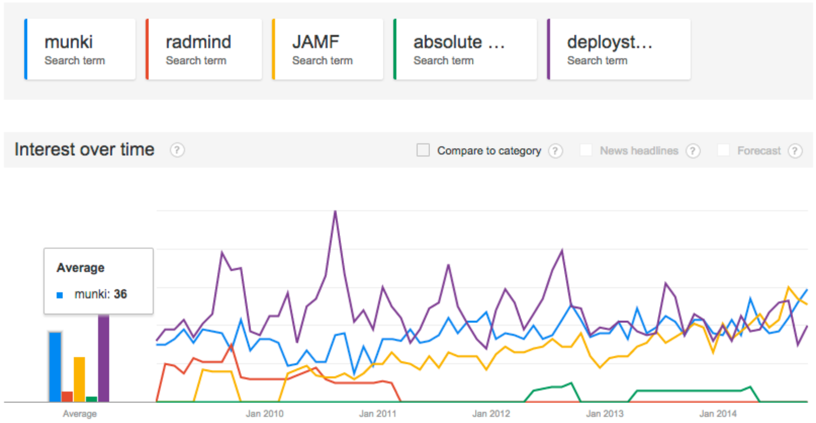
Autpkg even can take a cask directly from homebrew cask and use it to fetch a download with the BrewCaskInfoProvider.

trending

Changing gears for a second, when I was preparing this talk I decided to quickly check google trends for what it had seen in the way of traffic for various search terms,

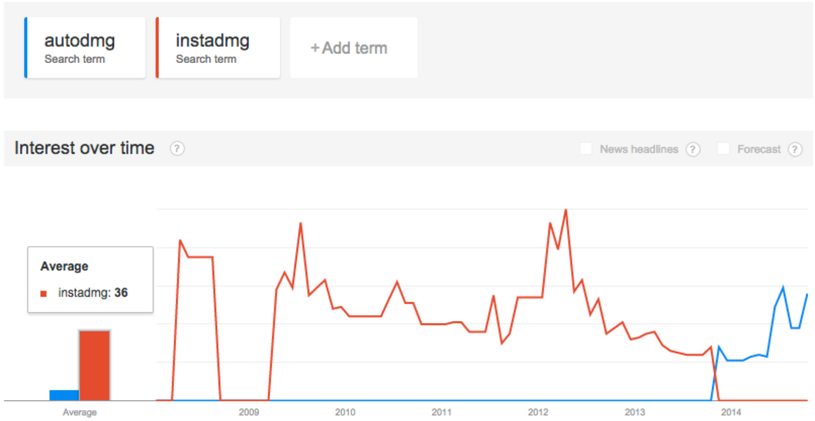


and here's autopkg's acscedency over the past year or so, in comparison to autodmg which actually came out much more recently



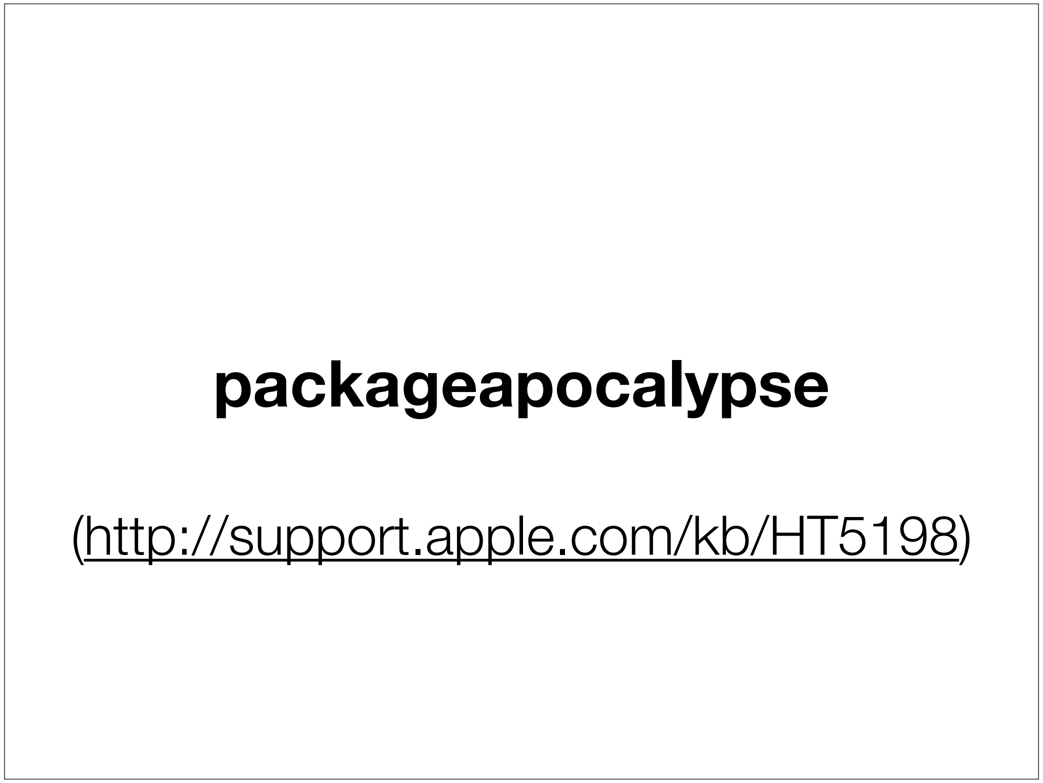
And then some comparisons to the free but closed source DeployStudio which is the purple line there, vs. companies with an actual marketing and PR initiative vs. munki - which is the blue line. And this is since 2009, refined to google's 'computers and electronics' category, so it's somewhat clumsy but you get a feel for relative activity, open source can hang in the midst of all of this.

And in the process of doing these quick queries I was reminded of something I was pretty involved with not so long ago,



instadmg.

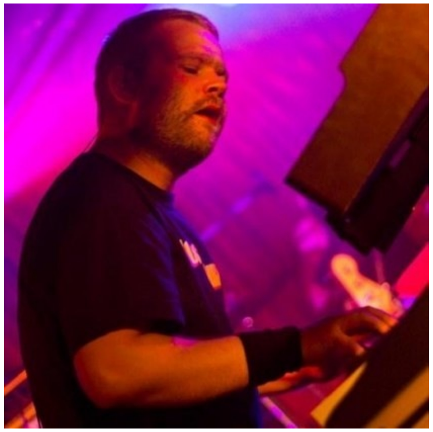
It has luckily been eclipsed by Autodmg, but several semi-derivative projects have carried over aspects of its design or the instaUp2Date companion script's in particular - one of which is checksumming the Apple patches fetched over the network as both an integrity check and you may even say, a security measure.



packageapocalypse

(http://support.apple.com/kb/HT5198)

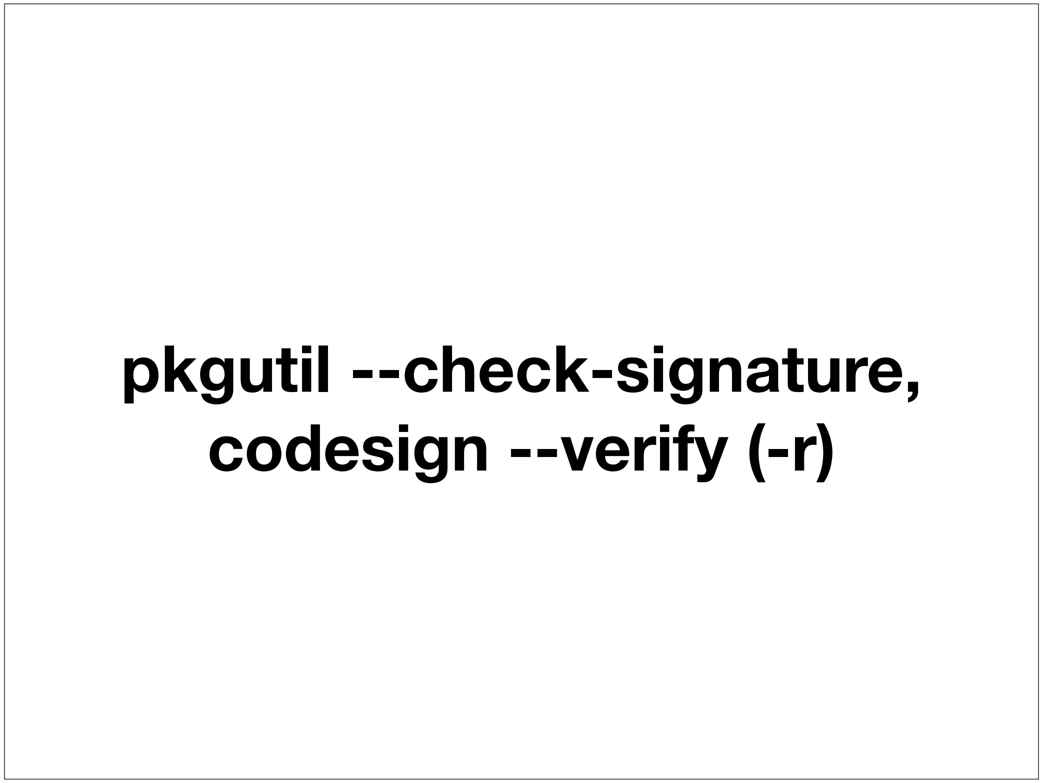
Now modern Apple, most notably coming to light around the time of March2012's PackageApocalypse, has tried to get a handle on validating what we install. And they've instituted various other things like xprotect, gatekeeper and practices like signature and certificate verification to do more of a defense in depth approach for the different attack vectors, with various degrees of effectiveness or success. Now you may be wondering why am I mentioning all this, and where does autopkg fit in?



Thanks to Han-nes Yuutilai nyen... wait, this doesn't look like someone you'd take seriously from a security perspective, how's about...

Yeah, here we go. You verify the certificate on that package or signature on that code

or he will come down here from Finland and do it FOR YOU. Where was I. Oh yeah, there is the CodeSignatureVerifier processor that he's contributed to core autopkg that allows you to verify the ID that an Apple Developer Program member would have attributed to them.



pkgutil --check-signature, codesign --verify (-r)

And that's trivially done on a package with pkgutil, or for standalone app bundles, you can verify the signature on the compiled application itself.

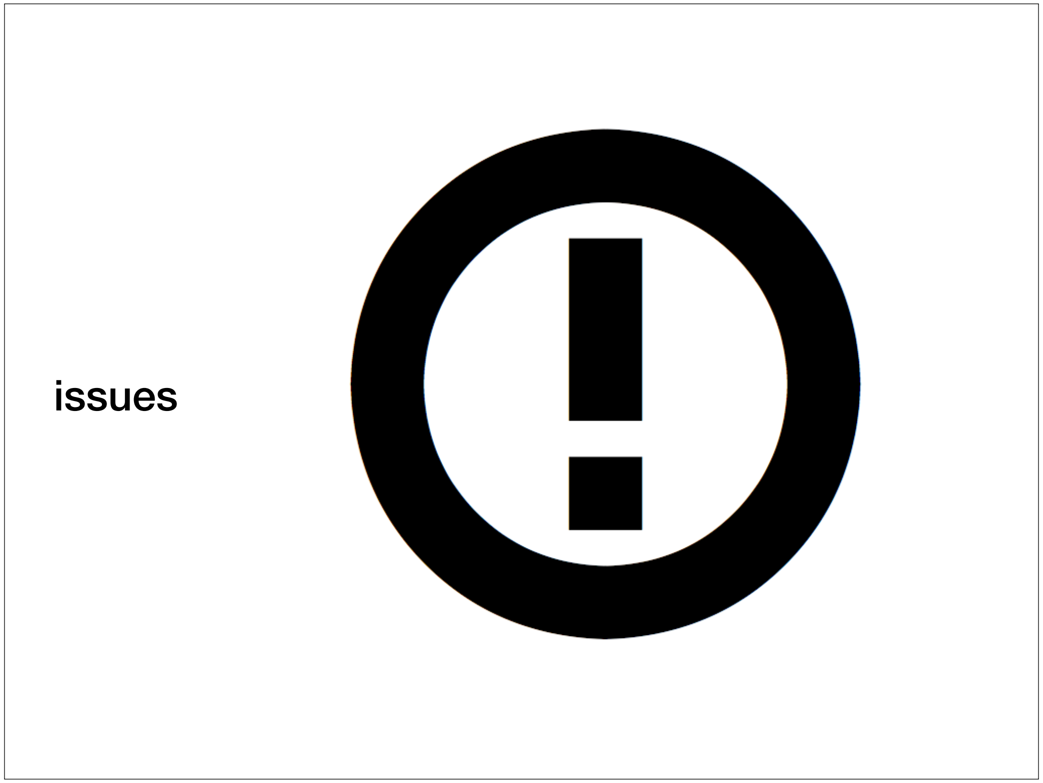
The latter actually requires 10.7 or greater to function as expected, and has a somewhat obtuse format as the output of the codesign command line tool, it kindof looks like the way you have a chain of trust with other PKI-related certificate-based systems. Why is this important? Well, perhaps your distribution system doesn't have the capability of verifying software once you've loaded packages in, and you're immediately making updates live to testers that are on the production network.

You don't want to swiftly push out a compromised update, especially if you've fetched it from the http URL of some random ContentDistributionNetwork. And under the covers, the way that autopkg relies on the python libraries that ship by default with OSX, is it unfortunately doesn't verify even https sites as I understand it, anyway...

Now I know what you're saying, way over my head and insert I don't always test my code meme here. But I'm going to ignore that you're saying that and go into call to action 1:

take action!

if you have the choice between recipes to use, look for ones that leverage the codesignatureverifier processor.

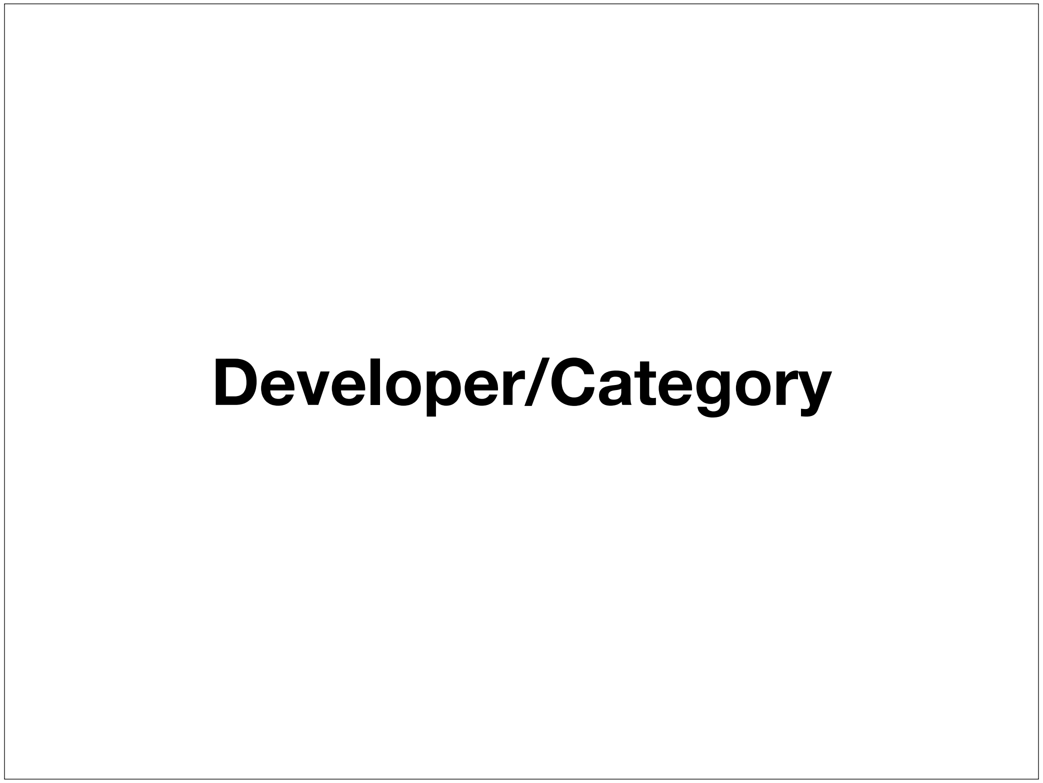


issues

This is another thing you can helpfully file an issue about, because I'd really like to see more of the recipes I use taking advantage of this. Tim talked about contributing to open source projects in his great presentation earlier in the conference, and I'm hopefully going to help him hammer that idea home with a workflow you can follow to get some muscle memory. It's a wonderful segue into the next section,

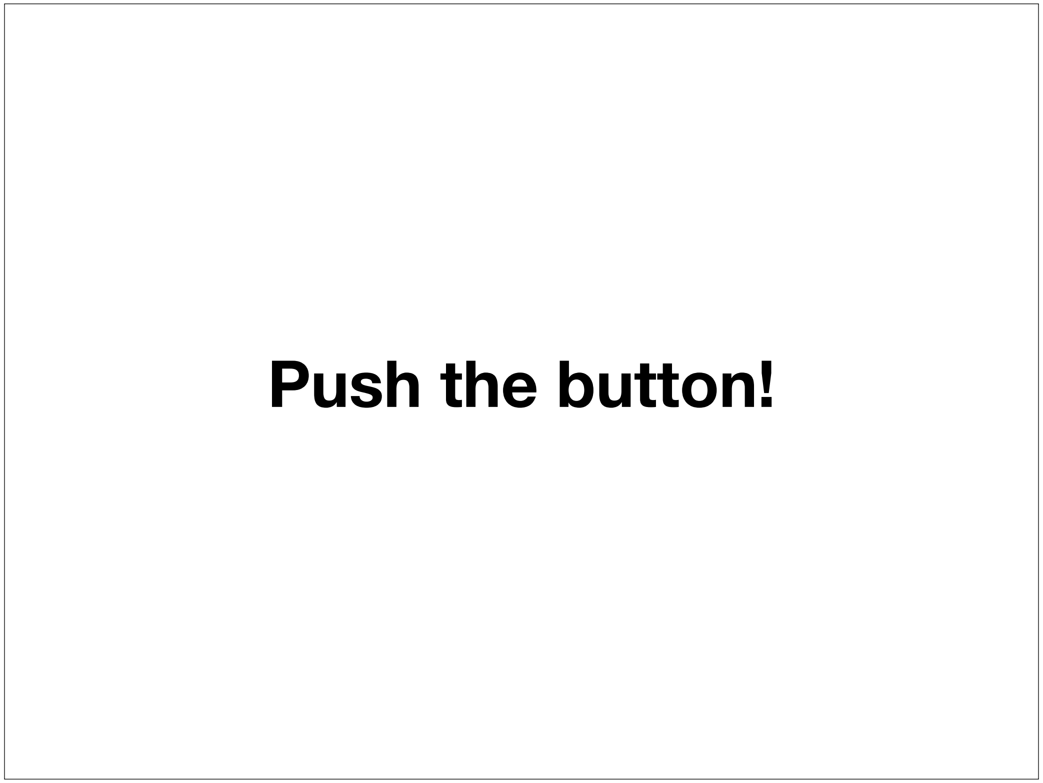
- **Refresher**
- **Out-of-the-Box**
- **Easy wins**

Ways in which even programming-averse folks can help out and we all get to experience some easy wins. Trying things out, giving feedback and/or filing an issue is the lifeblood of any open source project.
I also have the stage at this venue and can hopefully remind folks as Greg told you yesterday, Munki2 has been released for a while, and with it there's the more spiffy, Mac AppStore mimicing interface which, along with the tabbed flashy view



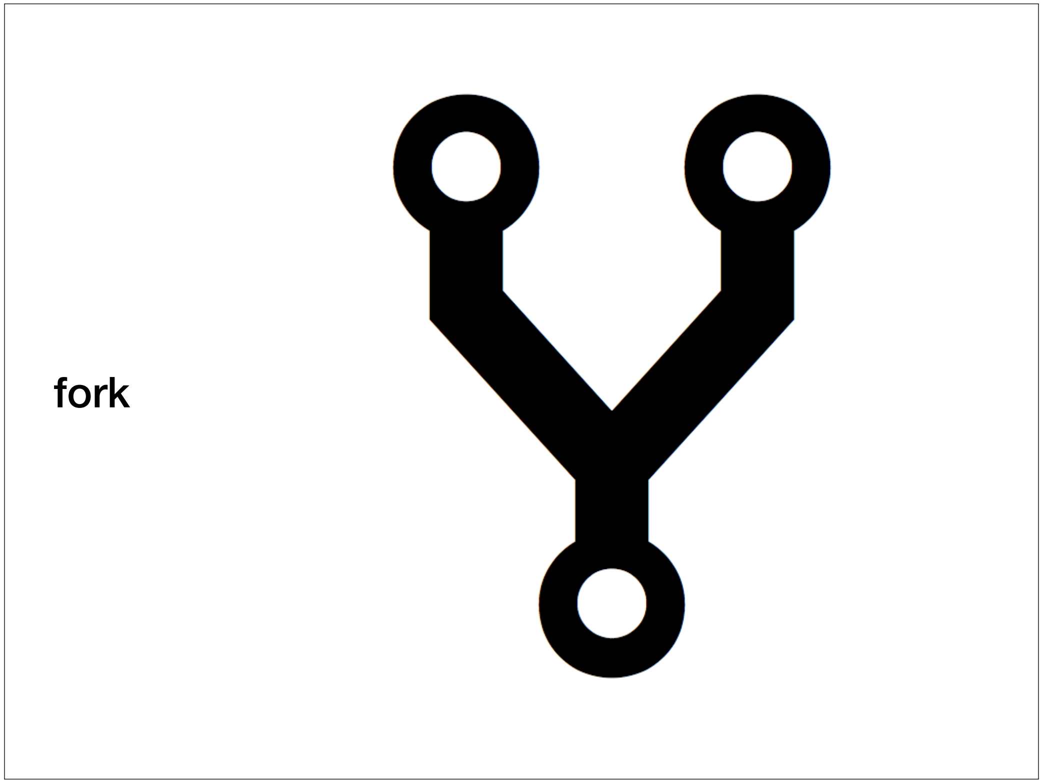
Developer/Category

has groupings by Developer or Category, which are direct analogues to the MacAppStore and constitute our second call to action:



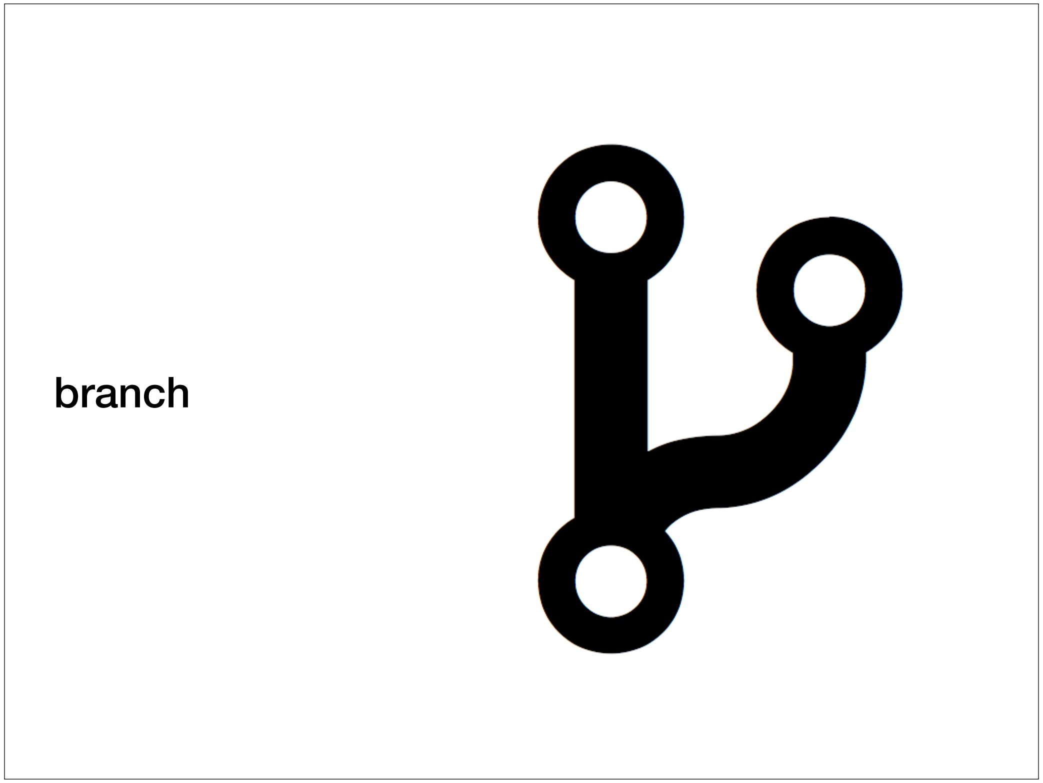
Push the button!

you can help all of the folks who's recipes you use by filing issues to have them add Developer and Category metadata for the munki recipes. But what if you're braver than that? You can say 'fork that person' who so kindly made their recipes publicly available,



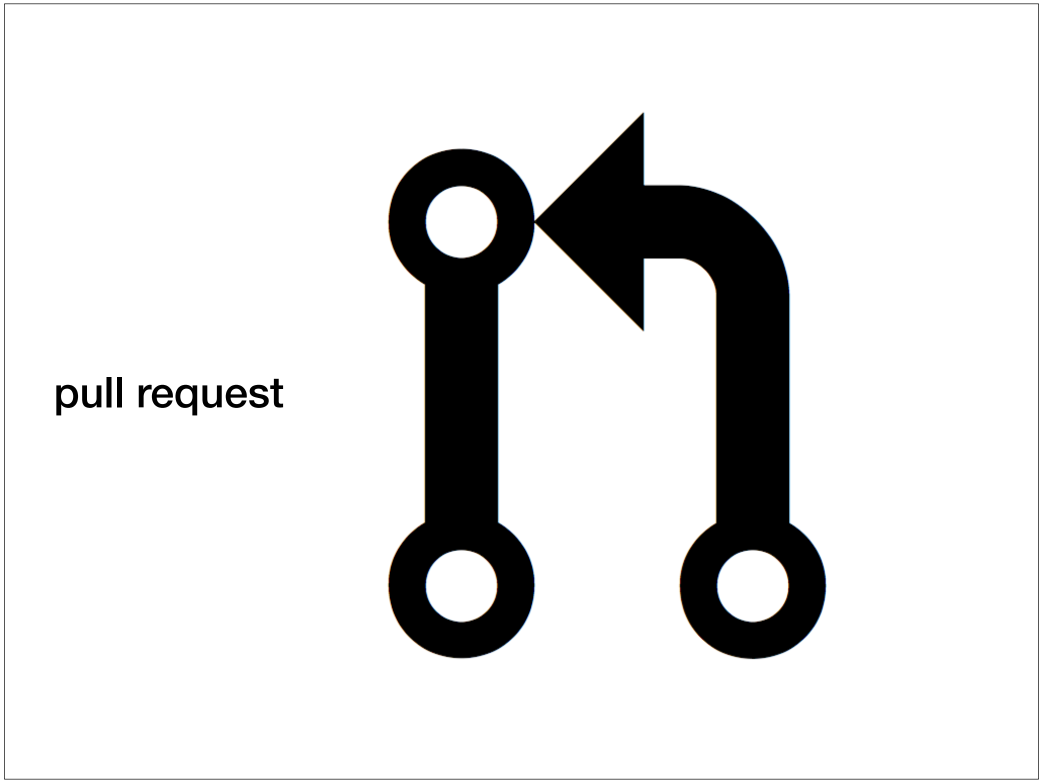
fork

and by clicking that Fork button when on the github webpage where the recipes you use are,



branch

you can further, create a branch to address that missing metadata



pull request

and send a pull request to them once you're done, basically starting a conversation with them which is much easier to have when most of the work is already at least attempted.

If all goes well and your changes are merged, anyone else who uses those recipes would only have to run a repo-update to take advantage of the new info.

Get involved!

Get in the game! To paraphrase public enemy, go get it, get involved, because my peoples in the street are willing to work it out. Hopefully I'll have an article coming out in MacTech with that exact workflow documented, and as you all get subscriptions as a perk of attending MacTechConf, you can read it at your leisure.

##osx-server

Or, stalk me or any of the other folks in ##osx-server IRC who are friendly and helpful and totally not nearly as snarky as I am, we'll do our best to find the time to look over your shoulder if it's your first time working with git and text like this. I don't even call it code, all you have to keep in mind is only perfect spellers may enter this system. Speaking of IRC, in ##osx-server you have folks using all different kinds of tools and from all different organizations. So it isn't just a munki lovefest, we take all comers, as they say. The point of which is even if it seems certain mgmt systems are making advances with integrating autopkg at a faster rate than the one you're currently stuck with, there are things Greg brought up at MacSysAdmin when discussing integrating new functionality via processors.

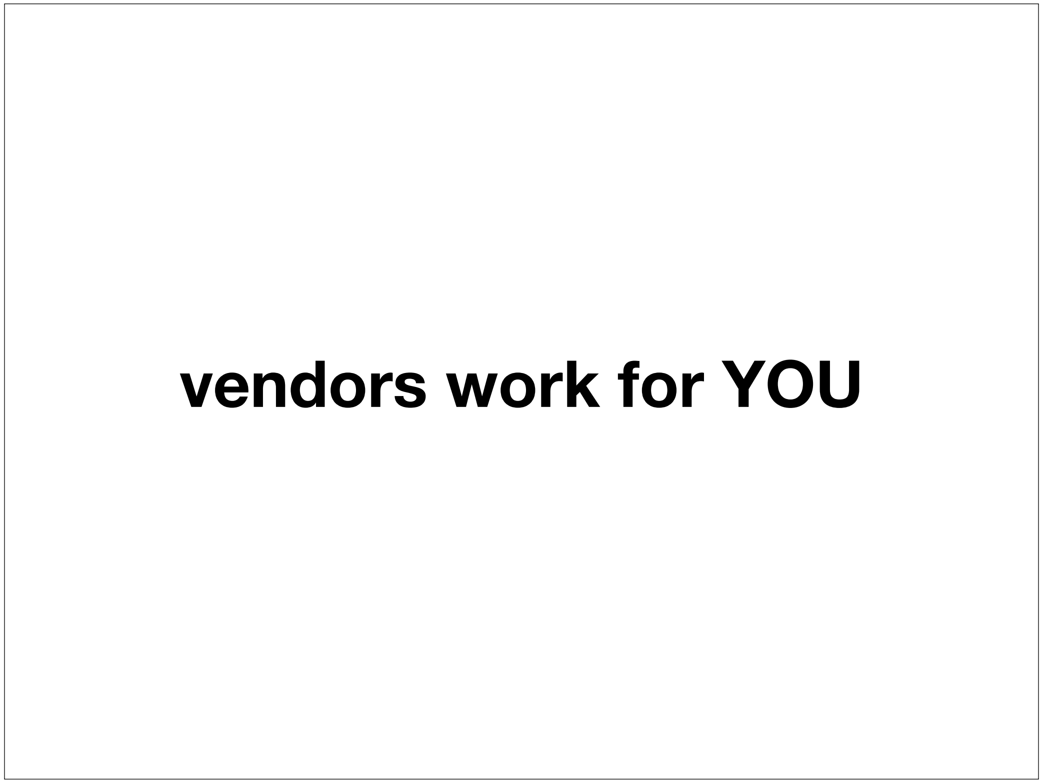
namespacing

And that is that there is an easy way of namespacing a processor to a given repo so autopkg can find and load it reliably while running any other random recipe that wants to leverage its functionality without it being built in to 'core' autopkg. A great example of that is the GitHubReleases processor Tim has in his own repo, so if the software is using a tagged, posted release on GitHub, you can leverage that processor to grab it even though it isn't in the core autopkg library.

When I made the jss-autopkg-addon some of you may have heard of, or even use, I was not following this way to more easily add that functionality, but now it's here and a supportable, discoverable feature.

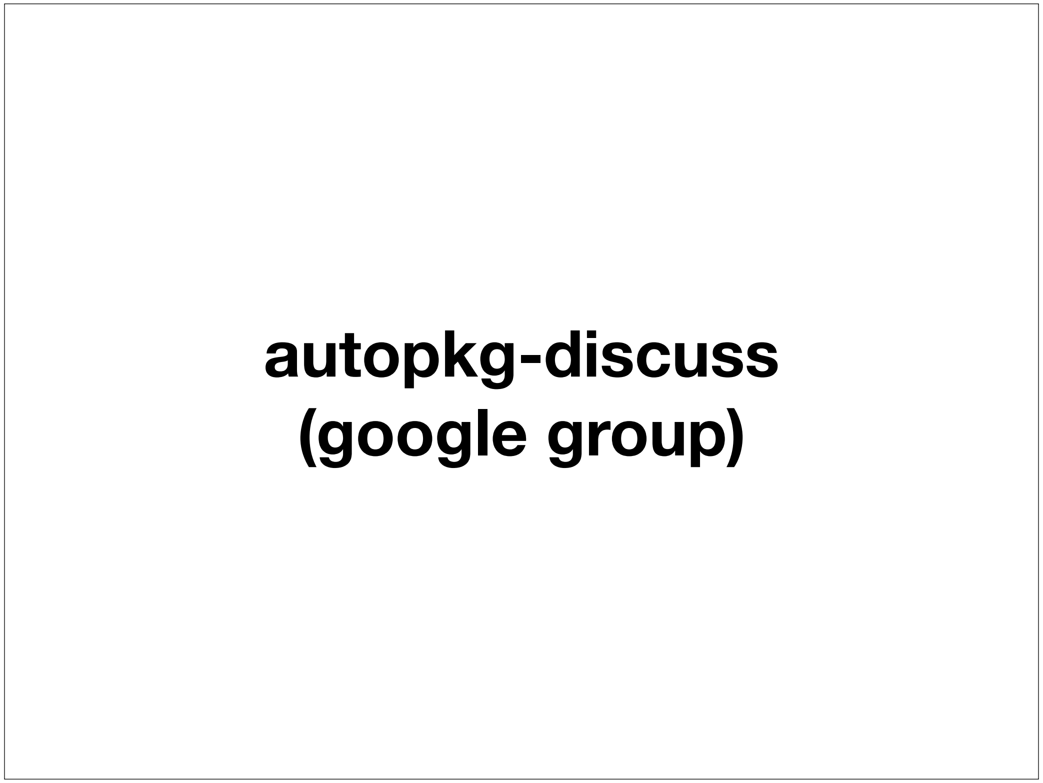
autopkgr+jssImporter

As a side note, I don't really think folks should continue to use my original jss-autopkg-addon, Autopkgr is smartly using Shea Craigs version which is much more capable at making the JSS really sing, he just released version 4 which supports multiple JDSes, good stuff



vendors work for YOU

To bring this section to a close, the vendors you're trying to strong-arm in order to allow autopkg to integrate with it, should be aware of how open our community is to helping them. Watchman Monitoring, for example, have even gone so far as to write a recipe for their own product, so some folks are doing the right thing out there. My suggestion is to approach your vendors



autopkg-discuss (google group)

and feel free to ask on the autopkg list if someone is already tackling the integration. You may even nerd-snipe some code whisperers to charm them pythons for you.

For the rest of us mere mortals, there's still hope besides reading wiki pages and the back issues of MacTech where Greg introduced autopkg,

recipeGenerationUtils

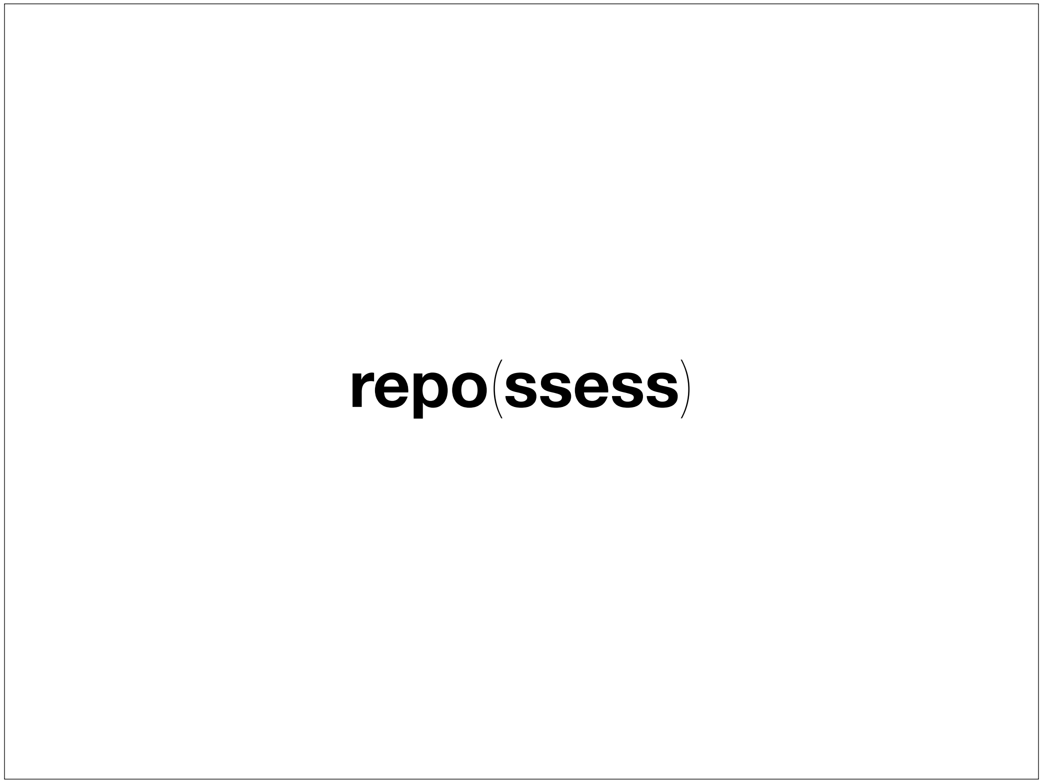
and that is the recipeGenerationUtility scripts I wrote when starting out, and will hopefully be updating as the best-practices, state-or-art of making autopkg recipes evolves over time.

As you may have seen when I announced this and reffered to it in a couple of blog posts, you feed a few pieces of information into a python dictionary and it spits out recipes. This is a great example of the true spirit of sysadminery, which is to be as lazy as possible.

I'm looking at incorporating CodeSignatures into it as part of the download step, and hopefully for the folks who don't think the wiki is quite instructive enough on making recipes yet, I hope it can be seen as a good example of what goes where. If I do say so myself.

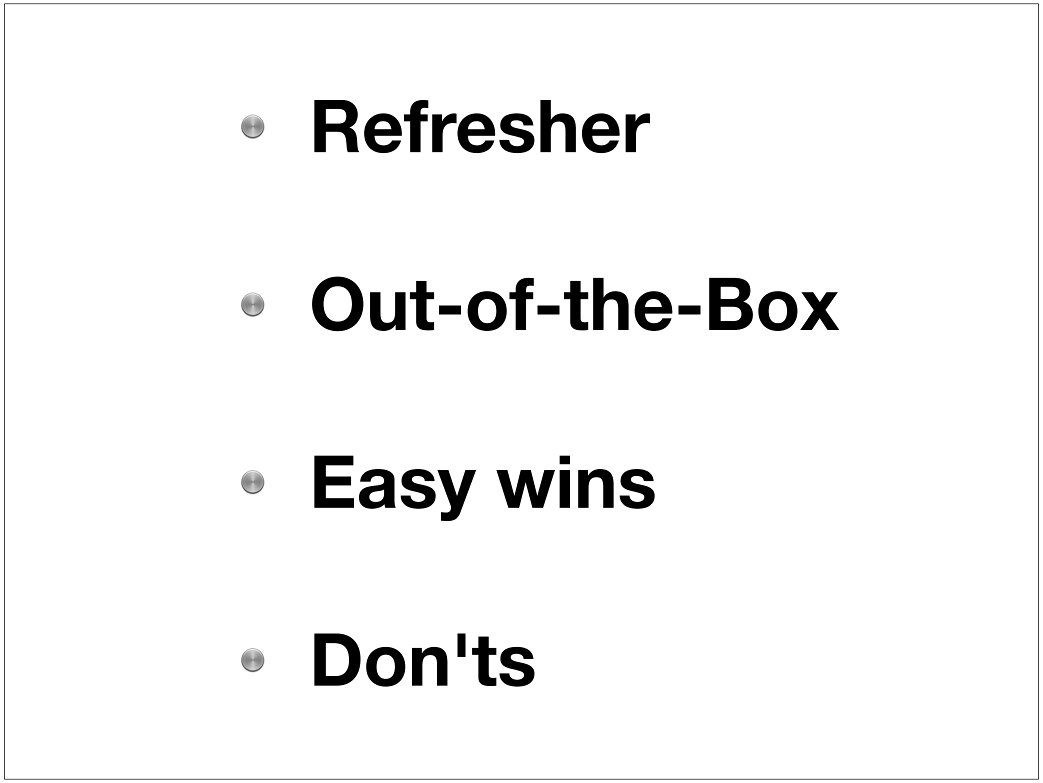
more labor
via automation…

As a stress test for these scripts, I even generated recipes for over 300 pieces of software from the homebrew/cask project I mentioned earlier… mostly because I didn't realize Tim had already made a processor for that exact task… I may or may not have been close to that xkcd cartoon about the threshold at which you're over-exerting for the sake of automation. For the curious, I threw them in a repo called Experimental-Brew-Cask-Recipes on my github. But this brings up another place where the less-experienced can pitch in, because that's a bunch of munki recipes that I couldn't make individual decisions about while running the script. We mentioned that recipes made for munki1 wouldn't have Developer and Category info, but mass-produced recipes from homebrew/casks also didn't get descriptions to show as a blurb when your customers see the optional install in ManagedSoftwareCenter.



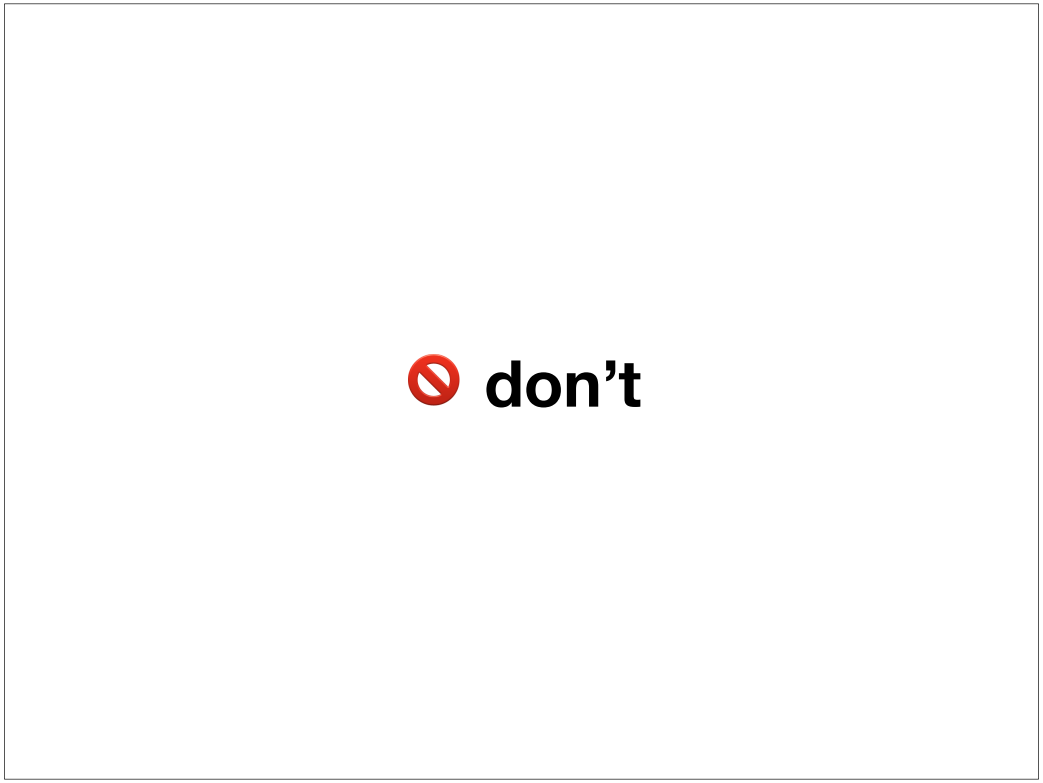
repo(ssess)

Call to action #3, take any of those cask recipes that you find useful and spiffy them up, inserting the copy for a blurb about the product or just otherwise verify it works for you, push a repo of the ones you want to github, and BOOM! You too can get into open source!



- Refresher
- Out-of-the-Box
- Easy wins
- Don'ts

And on to the last section, these have all been relatively conquered areas I've discussed so far, but those who know me may reasonably refer to me as an enthusiast. And so I try stuff! And sometimes it's well intentioned but doesn't work. And then Greg has to slowly, repeatedly explain to me why the things I think we should have won't actually work.



🚫 don’t

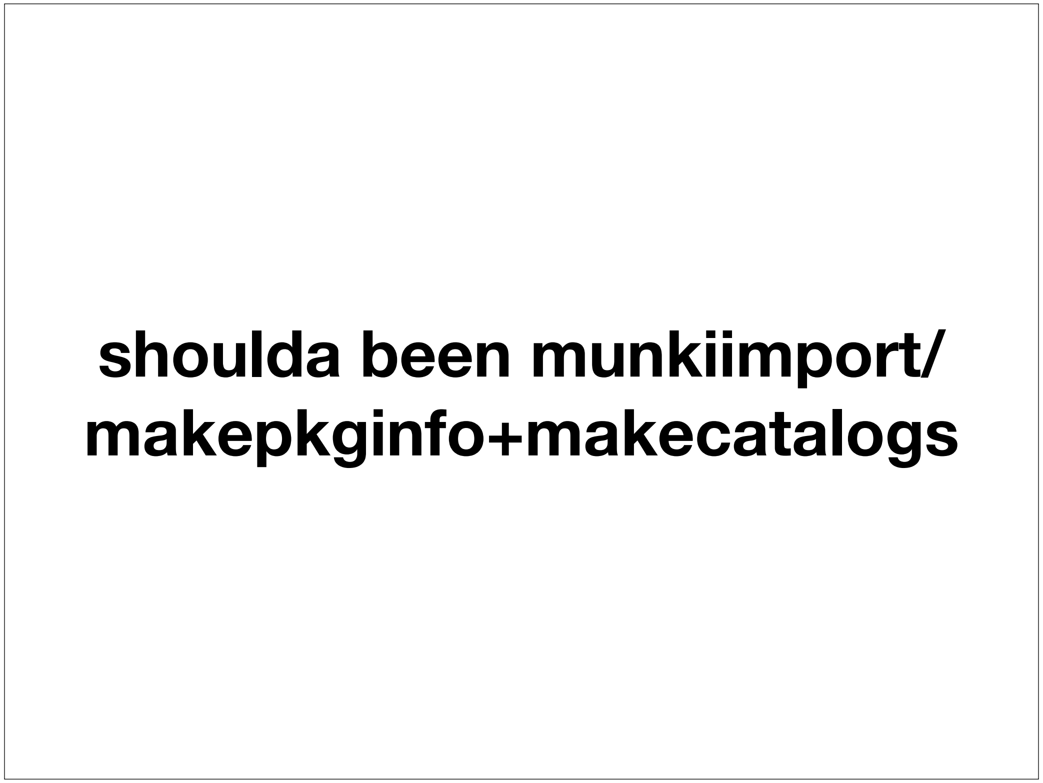
I'm saying this so when you see me being critical of how other systems work, know I hold my practices up to the same scrutiny, but to get back on track, here's what not to do when it comes to autopkg:

‘autoPromoter’

I wrote a processor specifically for use with munki, it looked for when the software had been imported into the repo, and did a little date math and, if a certain configurable amount of time had passed, it would 'promote' the update to the next, wider testing group. I got the idea from how Google's munki-server solution called Simian treats Apples updates,

and I even went so far as to check what day of the week it was, to make sure I wasn't pushing anything live on a Friday or Monday.

The problem was it didn't belong in autopkg - the concept of changing the catalog, or grouping of customers that the product would therefore be available for, isn't irresponsible in and of itself, but I was going about it in the wrong way.



shoulda been munkiimport/ makepkginfo+makecatalogs

First, munkiimport and its makepkginfo backend should've been able to set this promotion date without autopkg having anything to do with it. Second, makecatalogs should've been the point at which the promotion actually occurred. I did get Greg's interest peaked for the briefest while, and he even saw how one could perhaps clear out or archive older and rarely-accessed product versions to keep catalog and repo size down. I do plan to revisit it, but this is a story to say that while there's a lot of potential to improve things within the autopkg model, this just wasn't the right fit with the intended goals of the project.

Another way I conflated automation with autopkg was the idea I had for a SUS importer,

‘SUSImporter’

also for use with munki, since it can set an enforcement date on an apple update even if it doesn't push the package from its repo, that stuff belongs in reposado anyway, all that's necessary to have this functionality work is to run the makepkginfo command with the apple-update metadata you want specified. Where the feature I wanted to build comes in is that I want a mechanism to sync the new software updates reposado pulls down as they appear, to start it on the testing track towards wider release. This is something else that has no place in autopkg, Tim's aamporter if you hadn't heard of it, that's for the adobe creative suite's updates, that is definitely MORE applicable, but is still a standalone project because it doesn't fit in with the operation of autopkg as a tool.

what's better than a million dollars?

What would fit, and maybe myself or someone in this audience could consider doing, are enhancements to processors like Anthony Rhymer's DeployStudio integration - right now it isn't sophisticated enough to update where in the workflow it refers to the package in question. I want that when bootstrapping imaged machines, as that's when I apply Munki2 to our Macs.

One last piece of inspiration I'd like to impart upon the willing among us with the impetus to contribute, is to take the example of another community member that we all have great respect for, Paul Suh. He also seems to be as enthusiastic as me, and likewise newer to python, but he's attempted to contribute to the processors that ship with autopkg with a longstanding pull request that perhaps needs more eyes on it, a compliment to Hannes Yuutilainen's CodeSignatureVerification task because Paul's integrated the signing of installation pkgs with a developer ID. When you chain that to Tim's process to check against a git repo for new releases, internal development can benefit from legitimately signed pkgs, so our customers don't at least have to deal with dialog-box acceptance fatigue. And it can happen as part of a continuous integration pipeline, like the real software devs.

Help stamp out alert message fatigue!

Sometimes I think Jenkins or the luggage will be subsumed by autopkg, but I'm just happy there's this much activity going on around the tasks that are difficult enough that they should be automated by those with a firm grasp of how to accomplish them, and everyone else benefits 'for free'. We should be accountable to however few of our customers there are that are sticklers with us about warning dialog boxes, BEFORE the security department gets involved. And efforts like this will help cure the plague of blindly clicking through warnings so hopefully our customers know they can trust us curating software for them.

final inspiration 💁

Final words: these past four years of autopkg have been very active and productive, but there are plenty of ways folks can help out with recipe enhancements and points of extension to be added. It feels great to be involved, and to the folks out here just taking an interest in it, I can't wait to hear your ideas and see your contributions.