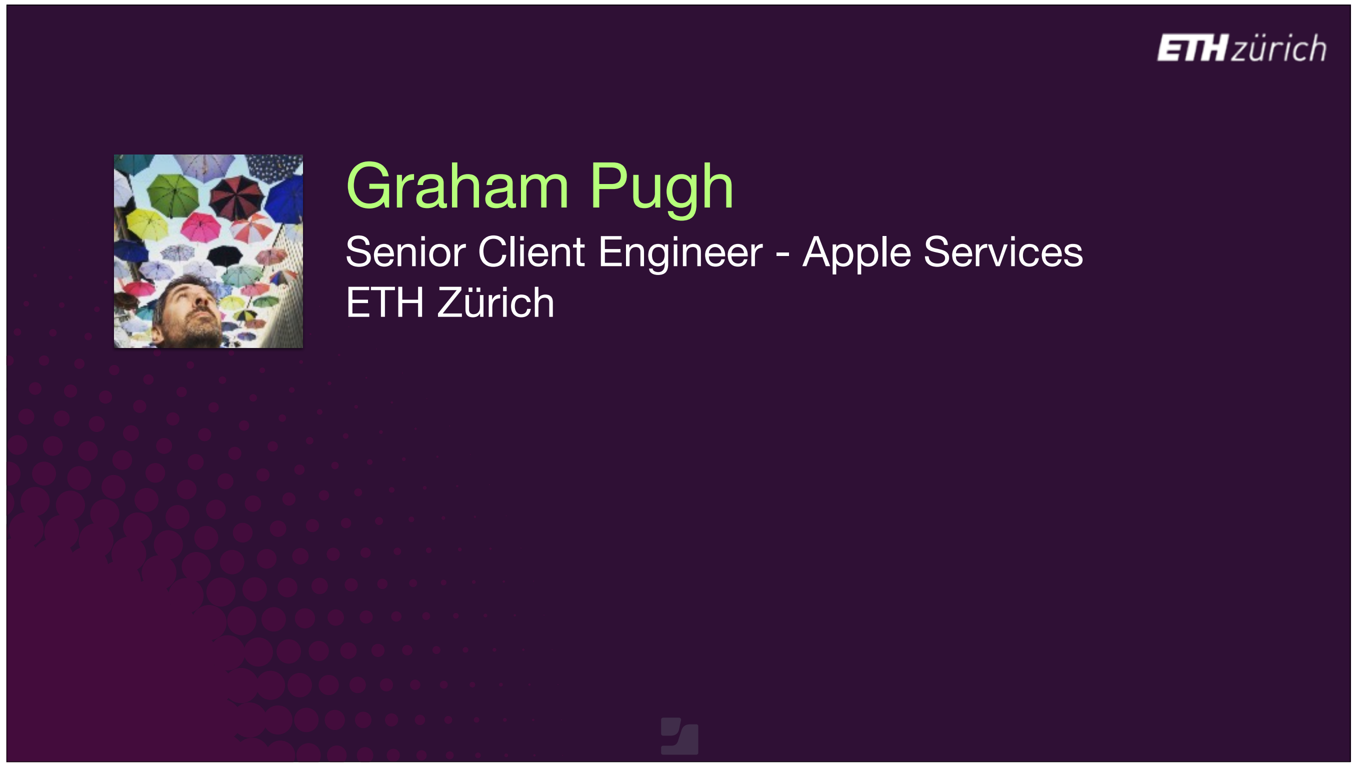
UP NEXT

Jamf Pro and AutoPkg

13:30 - 14:15

JNUC

2019



ETH zürich

Graham Pugh

Senior Client Engineer - Apple Services
ETH Zürich

My name is Graham Pugh and I currently work in Switzerland at ETH Zürich as a Senior Apple Client Engineer.

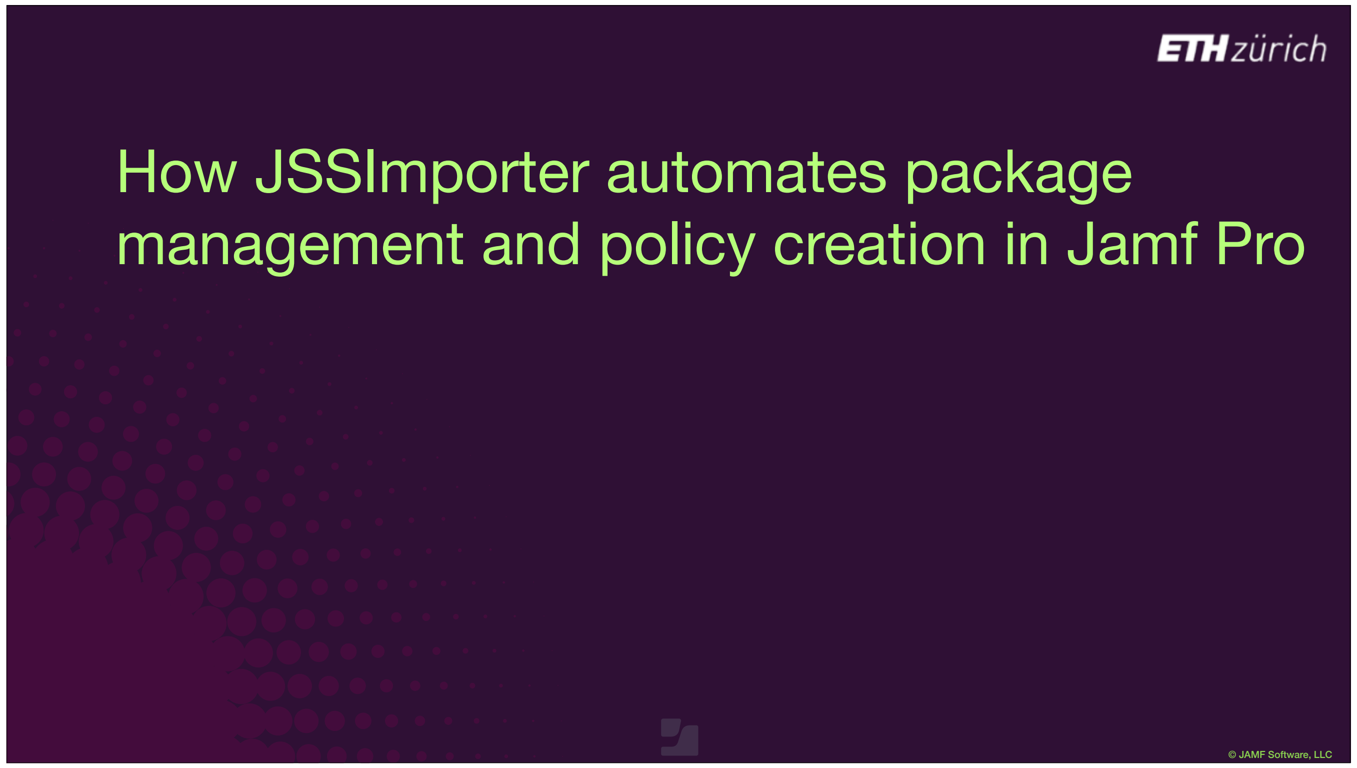
[click] These are my social thingies,
and I have posted links to everything I will talk about today in my blog at graham r Pugh . com, so if you hear anything useful during the session, there's no need to write it down. I'll also link to it again at the end.

ETH zürich

Graham Pugh

Senior Client Engineer - Apple Services
ETH Zürich

@GrahamRPugh
@GrahamRPugh
@grahampugh
grahamrpugh.com



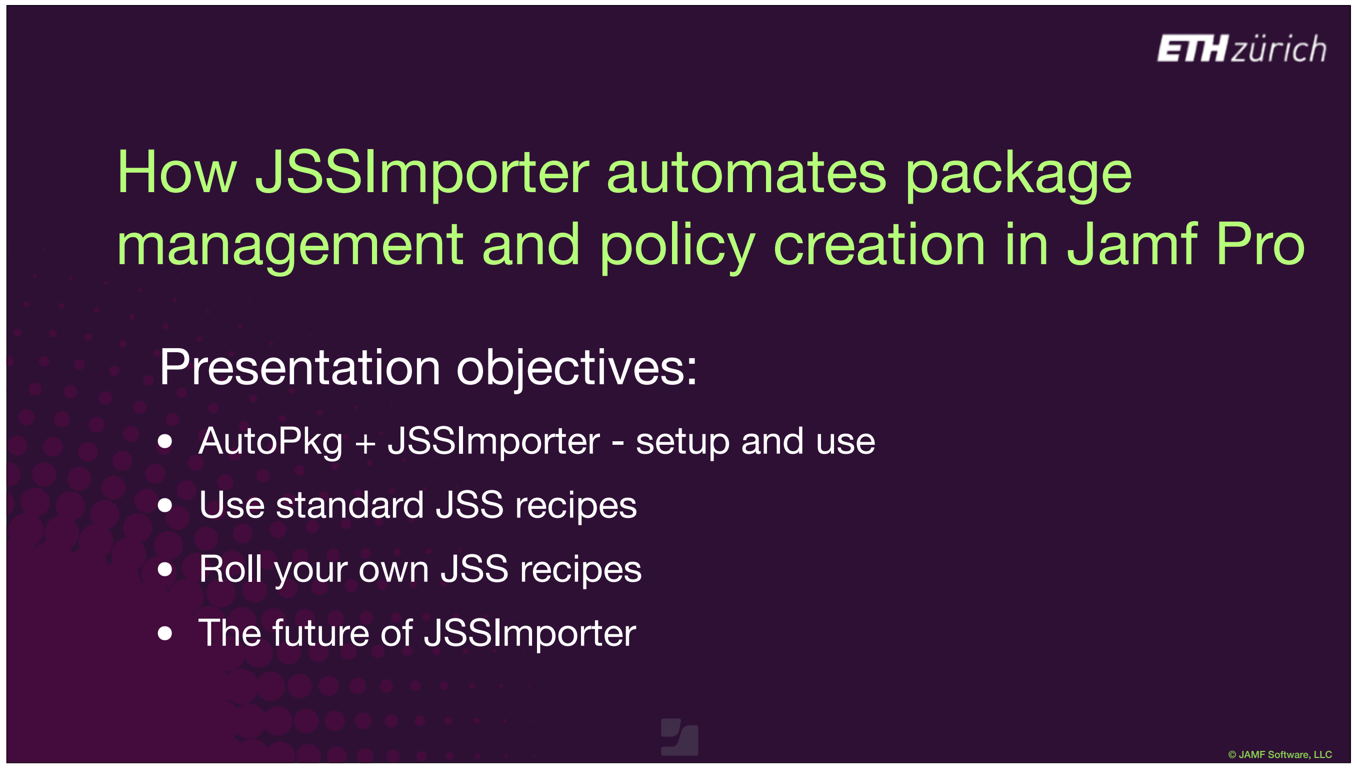
ETH zürich

How JSSImporter automates package management and policy creation in Jamf Pro

© JAMF Software, LLC

I'm very happy to talk today about using AutoPkg with Jamf Pro.

- Can I ask – How many of you are manually creating and importing packages into Jamf Pro? Hands up!
- How many of you would rather not do that manual work?
- How many of you are already using AutoPkg and JSSImporter?


ETH zürich

How JSSImporter automates package management and policy creation in Jamf Pro

Presentation objectives:

- AutoPkg + JSSImporter - setup and use
- Use standard JSS recipes
- Roll your own JSS recipes
- The future of JSSImporter

© JAMF Software, LLC

- Today I will try and cover everything from total beginner to advanced workflows.
- [click] We'll look at setting everything up and running AutoPkg recipes.
- We'll look inside recipes to help you understand what they do, and show what you can customise.
- Then look at use cases where you will need to create your own recipes.
- And finally, we'll wrap up with a look at the future of JSSImporter, and take questions.

- First, a little about me and where I work, and how that led to my involvement with JSSImporter.
- [click] The Swiss Federal Institute of Technology is one of the world's top universities, and the top in continental Europe.
- Like Switzerland itself, ETH has a very federal mindset, with a high degree of autonomy for professors and their institutes, who employ their own IT teams.
- The central ETH IT Services are a service provider to those teams.

ETH zürich

ETH Zürich

Eidgenössiche Technische Hochschule Zürich

Swiss Federal Institute of Technology

21,397 students

9,528 staff

528 professors

QS World University Rankings 2019 - 2020

1. MIT
2. Stanford
3. Harvard
4. Oxford
5. Caltech
6. ETH Zürich
7. Cambridge
8. UCL
9. Imperial
10. Chicago

Photo: ETH Zürich / Gian Marco Castelberg

ETH Zürich

Eidgenössiche Technische Hochschule Zürich

Swiss Federal Institute of Technology

21,397 students

9,528 staff

528 professors

Photo: ETH Zürich / Gian Marco Castelberg

ETH Zürich

Eidgenössiche Technische Hochschule Zürich

Swiss Federal Institute of Technology

21,397 students

9,528 staff

528 professors

Photo: ETH Zürich / Gian Marco Castelberg

- As those teams are independent of each other, the Apple Services team of Max Schlapfer, Kati Zehnder and I, with team leader Thomas Richter, provide individual Jamf Pro instances to each customer, just like an MSP.
- We currently maintain 31 on-premises Jamf Pro instances for our customers, and we have trained around 120 administrators how to use it.
- We provide over 100 software titles to those Jamf instances. That's not just the packages, but also the policies, smart groups and so on.

ETH Zürich

Eidgenössiche Technische Hochschule Zürich

Swiss Federal Institute of Technology

21,397 students

9,528 staff

528 professors

Apple @ ETH

3 client engineers

31 Jamf Pro instances

2000 managed clients

120 administrators

100 software titles

Photo: ETH Zürich / Gian Marco Castelberg

Our customers want to choose how to deploy the software, so we provide different policies for self service, auto-install, auto-update, uninstall and so on.

- We could not provide the level and variation of services we do without a high degree of automation.
- [click] To handle the volume of packages we generate and offer for deployment, AutoPkg is essential.
- And since we wanted to provide a single system for our customers to administrate their Apple devices, we didn't want to add a separate system such as Munki for package management.
- [click] Under these circumstances, JSSImporter is the only well established way to integrate Jamf Pro with AutoPkg.
- That's why I've got so involved with JSSImporter.

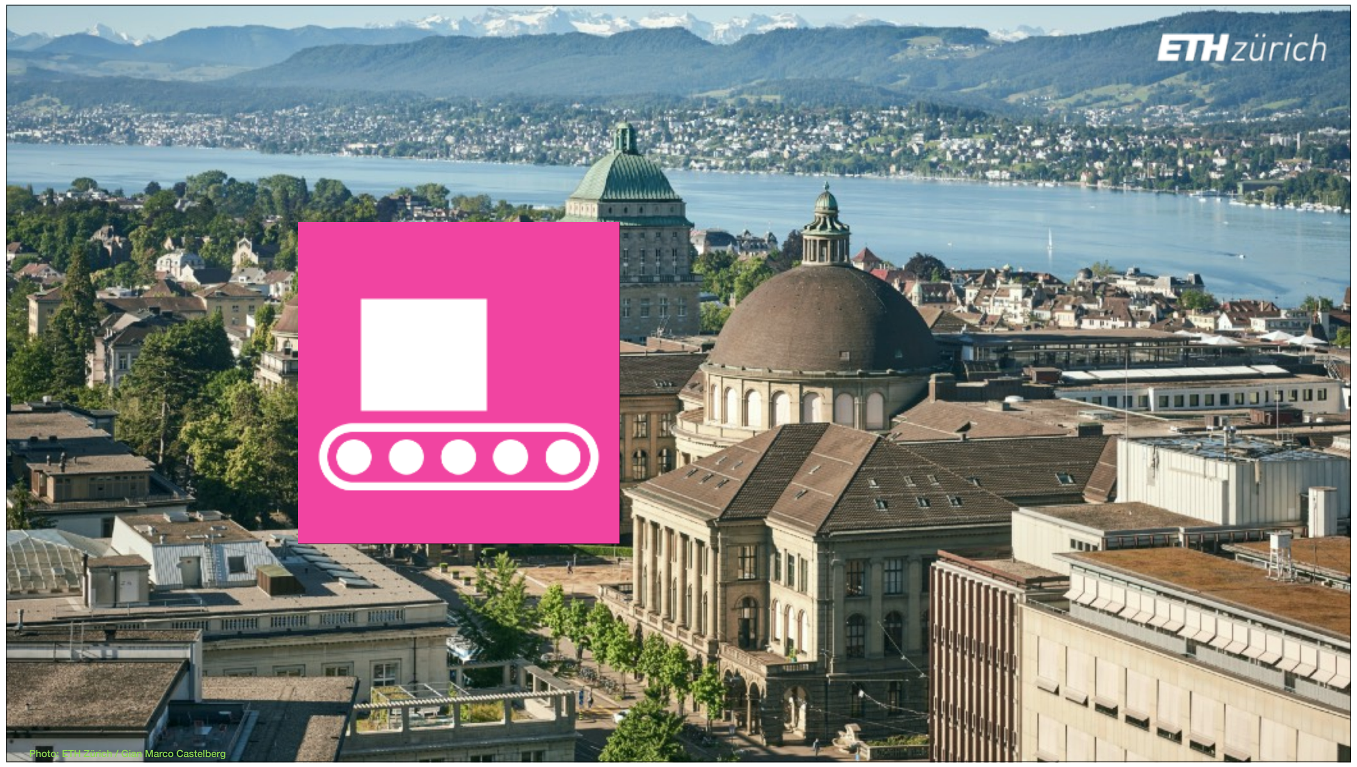


Photo: ETH Zürich / Gian Marco Castelberg

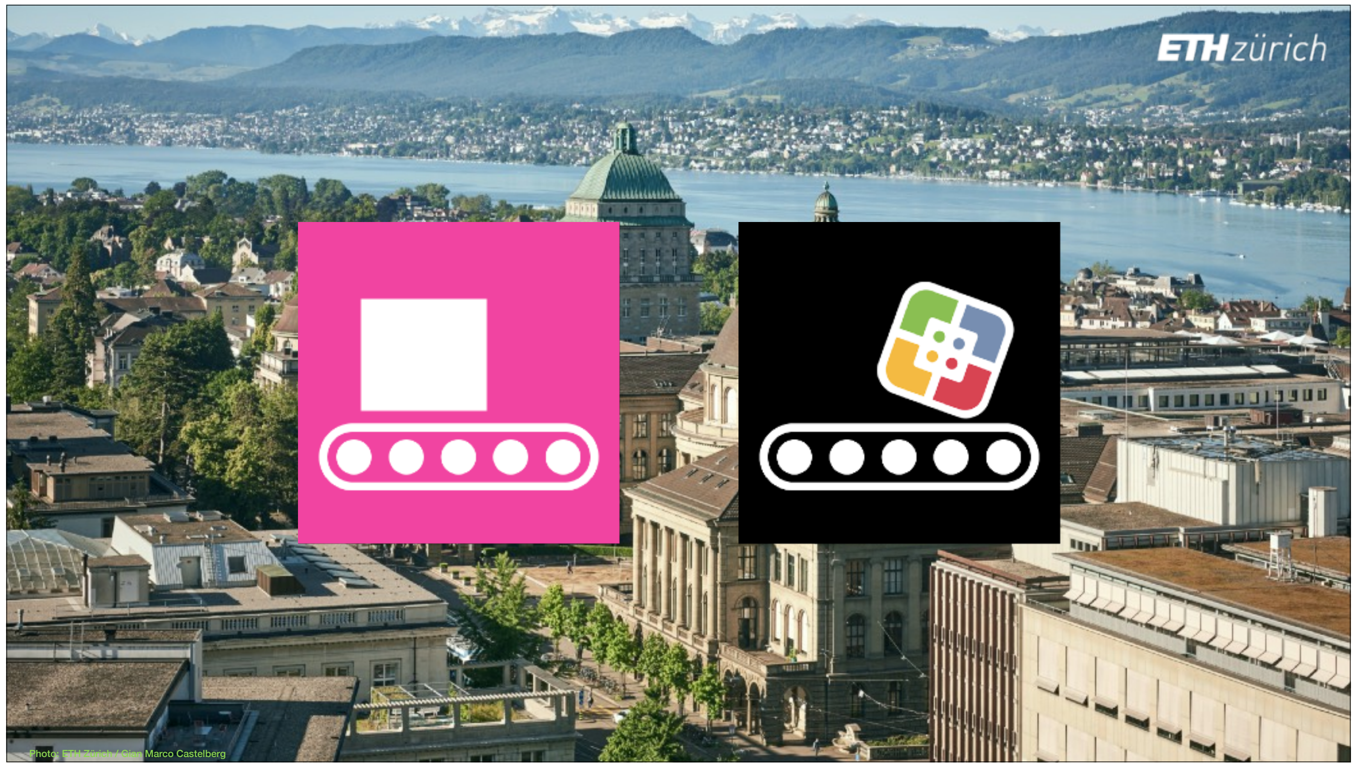
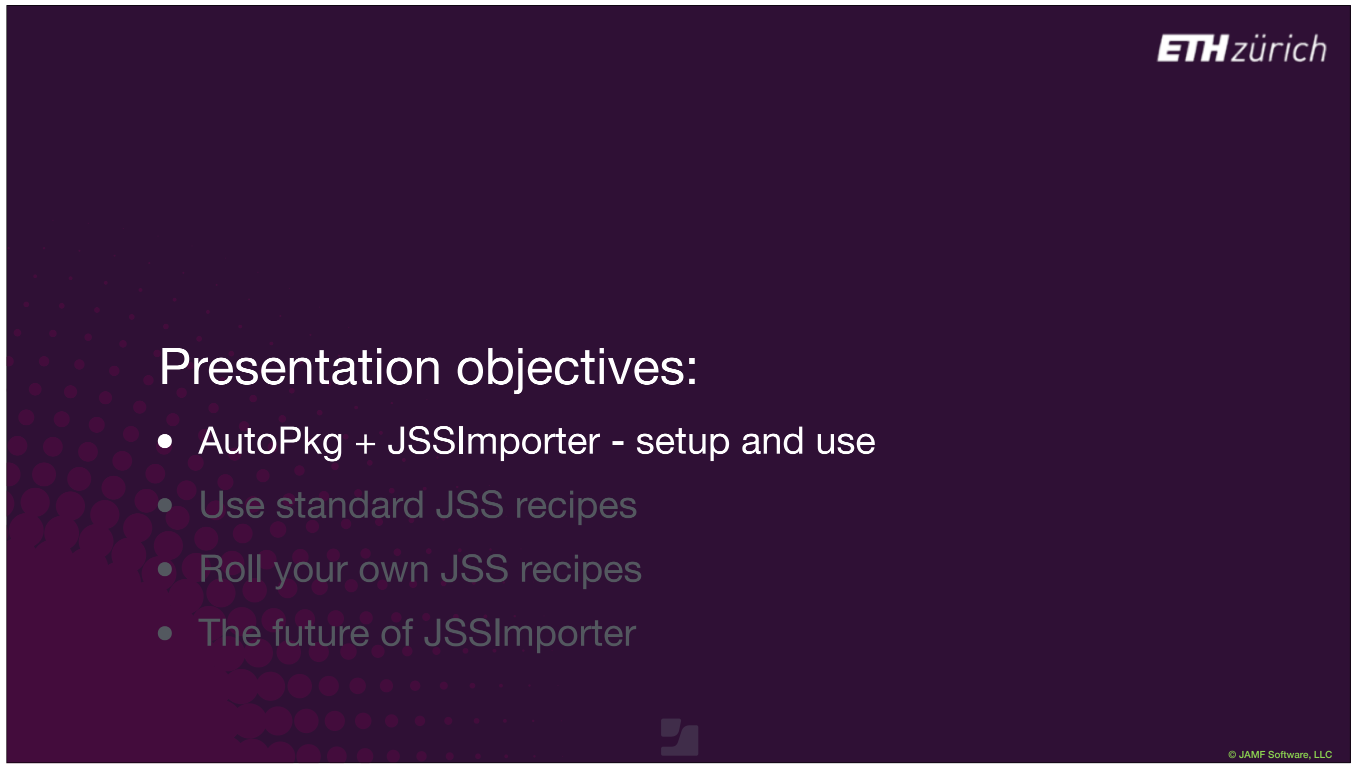


Photo: ETH Zürich / Gian Marco Castelberg



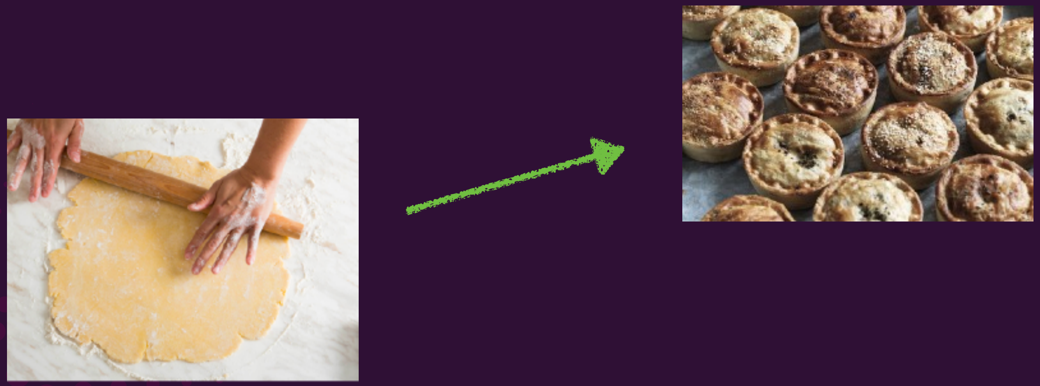
ETH zürich

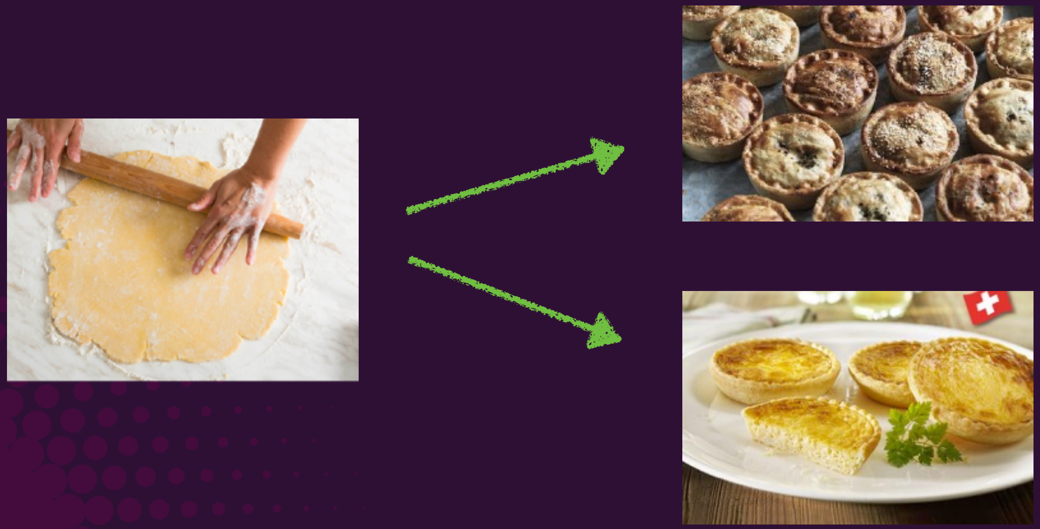
Presentation objectives:

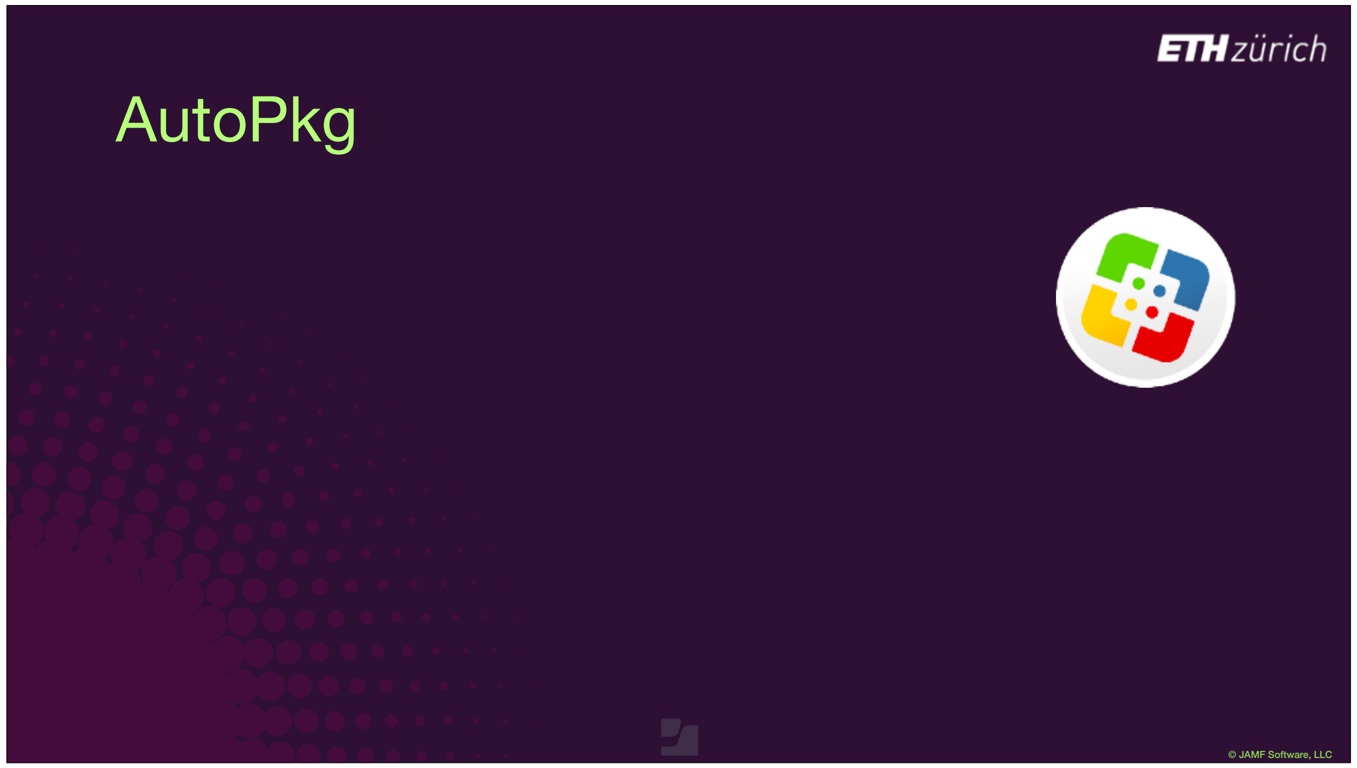
- AutoPkg + JSSImporter - setup and use
- Use standard JSS recipes
- Roll your own JSS recipes
- The future of JSSImporter

So let's dive in and look at what AutoPkg and JSSImporter are, and how they work.

- I'll start with an analogy.
- [click] As I'm British, I like to make meat pies.
- When I go to a meat pie recipe, the first thing it does is refer me to another recipe for making pastry.
- [click] Now I live in Switzerland, I might make Chäsechuechli instead.
- [click] The first part of the workflow is the same for both types of pie.







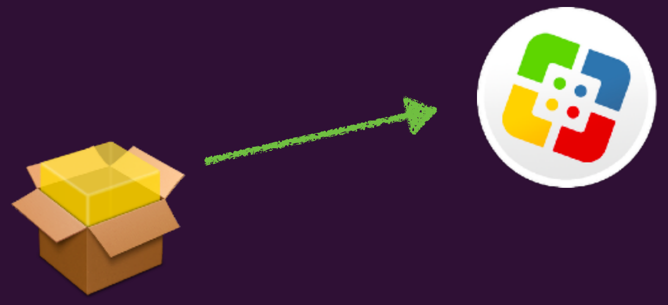
AutoPkg

AutoPkg follows the same concept.

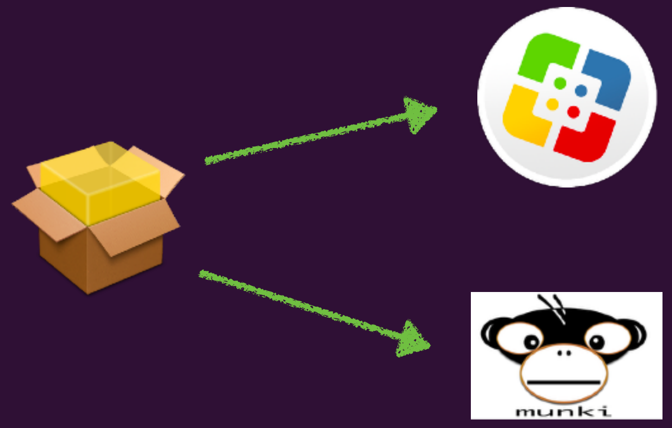
- If you want to get an application into Jamf Pro's repo.
- [click] You're first going to need to obtain a package, which might need to be built from a source DMG or a ZIP file.
- [click] That same package is used if you are uploading to any file repository, like a Munki repo.
- [click] Additionally, to get that source file, you need to know where to find it on the web, or wherever.
- AutoPkg uses different files containing instructions for each part of this workflow. These files are called recipes.

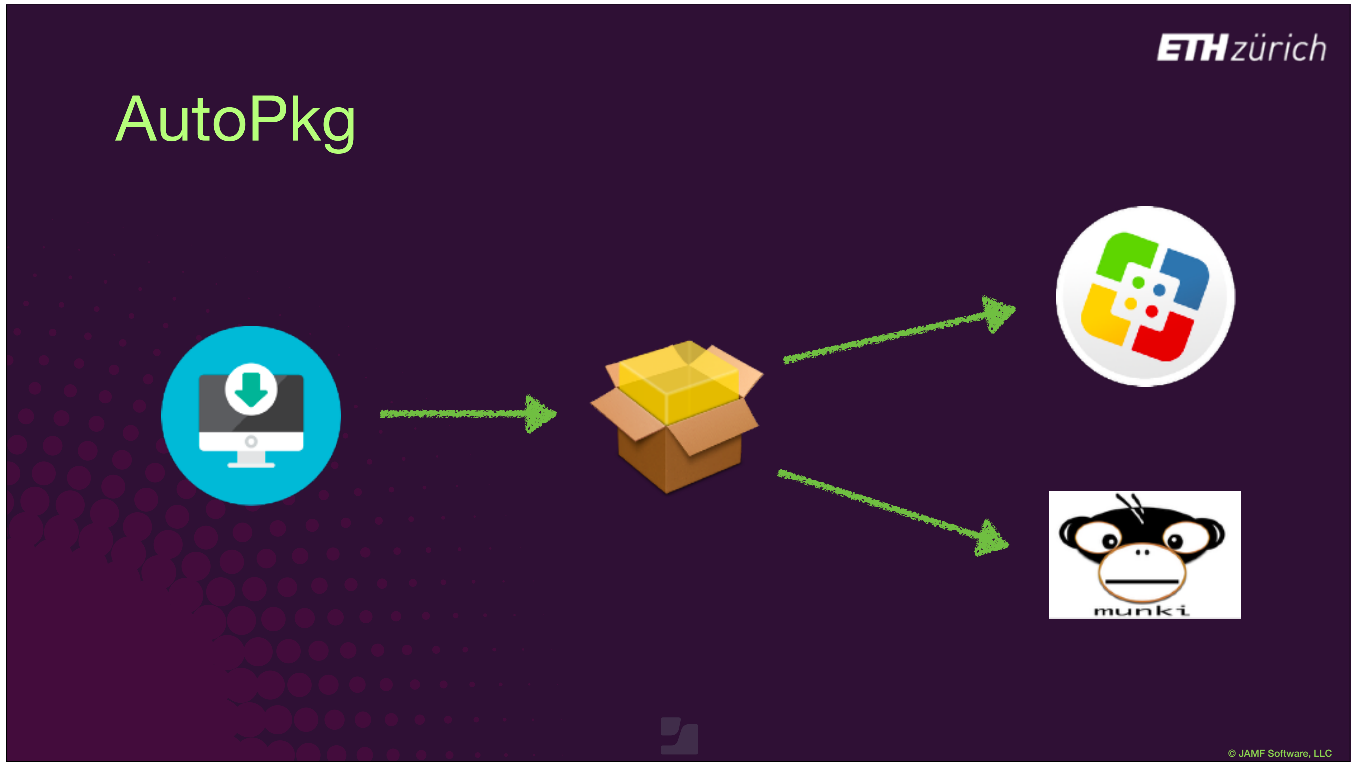
ETH zürich

AutoPkg



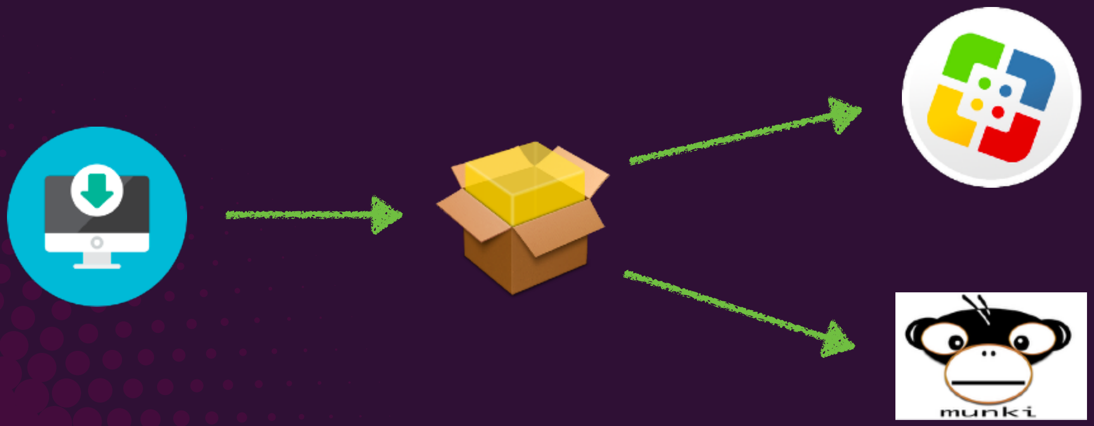
AutoPkg

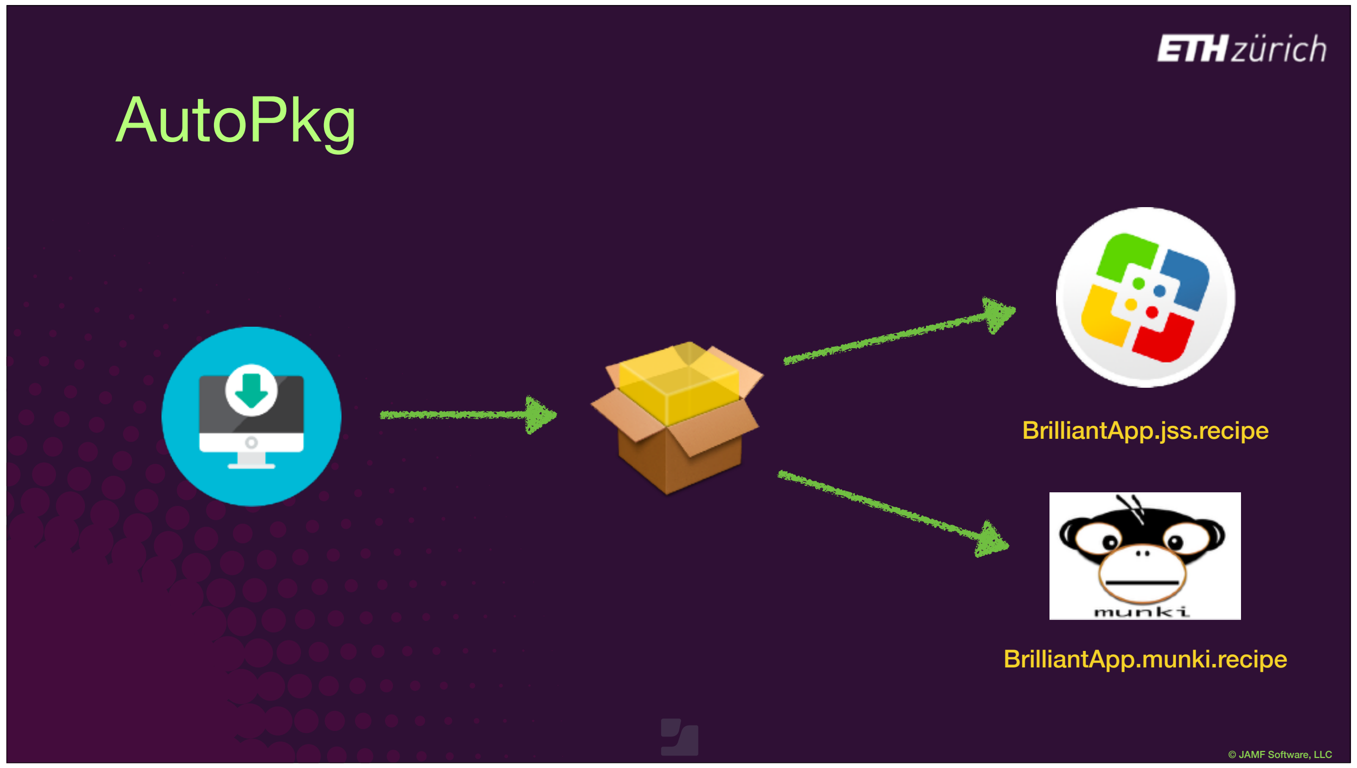




ETH zürich

AutoPkg





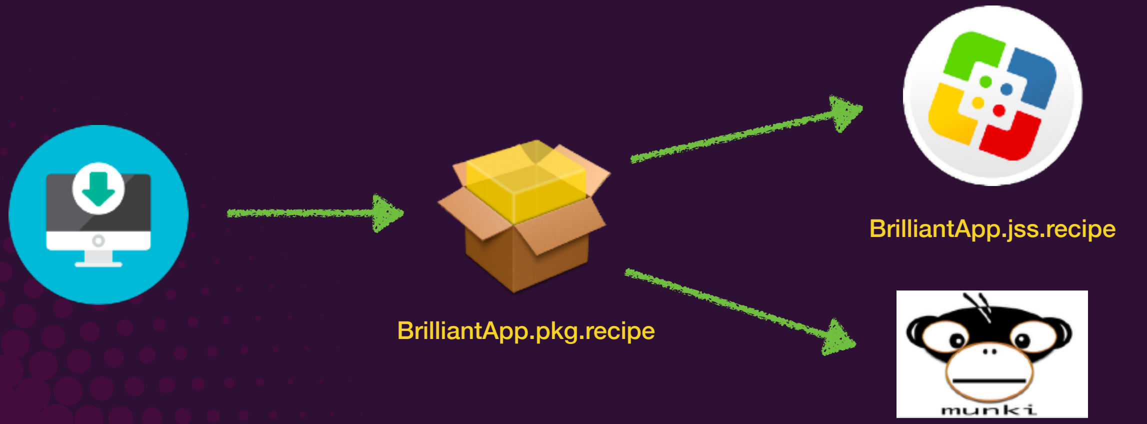
ETH zürich

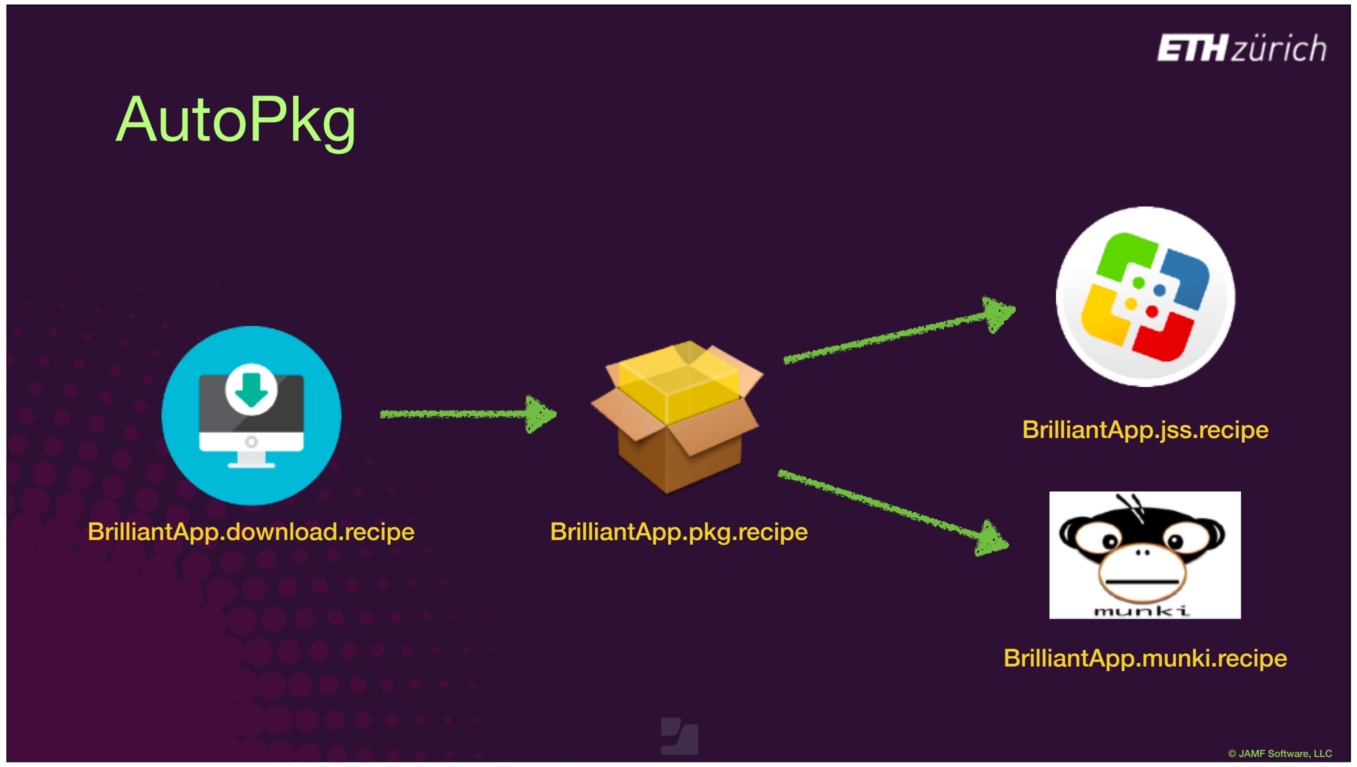
AutoPkg

- The JSS or MUNKI recipes tell AutoPkg how to upload the package to their respective repositories.
- [click] Each recipe also contains a reference identifier to the PKG recipe, referred to as a parent recipe.
- The PKG recipe contains the instructions to create a deployable package from downloaded source material.
- [click] The PKG recipe contains the identifier of a parent DOWNLOAD recipe that contains the instructions on how to obtain the installer in the first place.
- When you run a JSS recipe, it works through the parents first, so it can complete the workflow.

ETH zürich

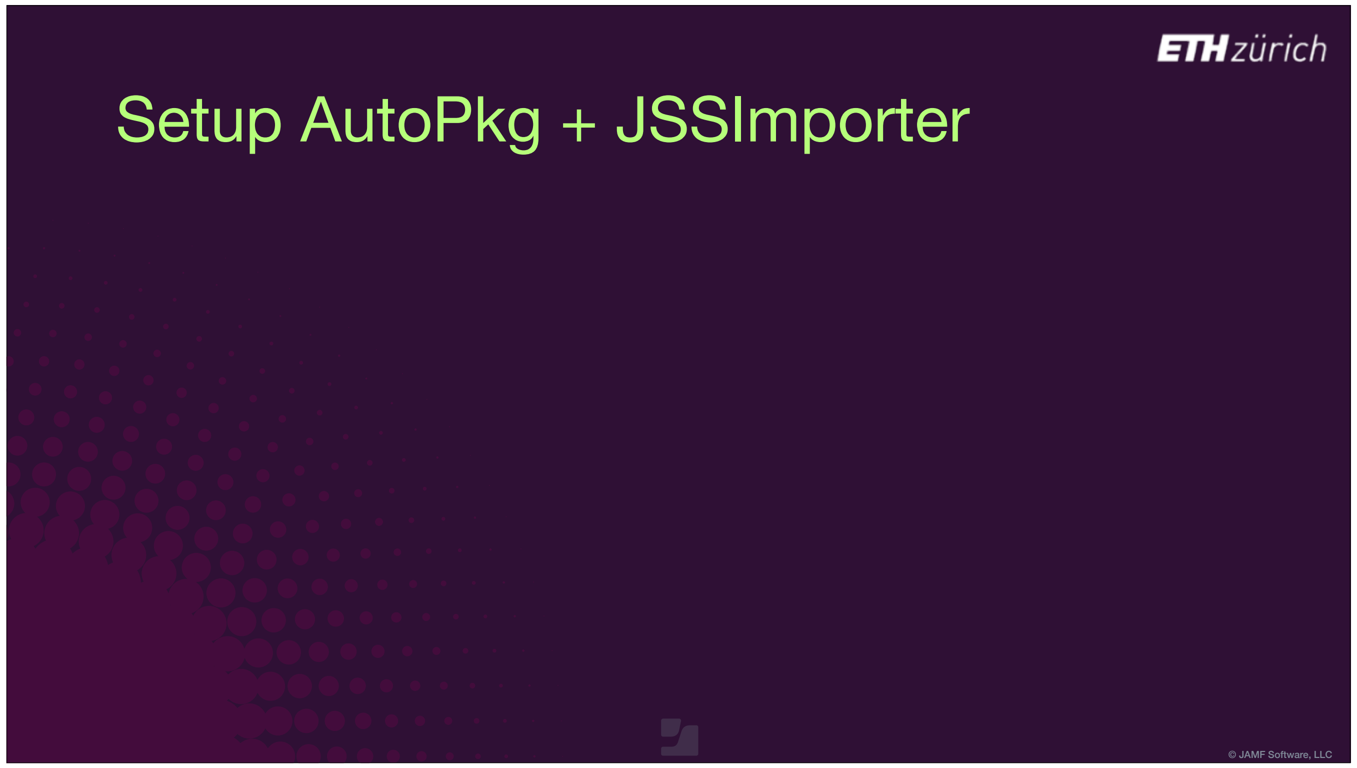
AutoPkg





ETH zürich

AutoPkg



Setup AutoPkg + JSSImporter

ETH zürich

© JAMF Software, LLC

To interpret AutoPkg recipes, you need to install the AutoPkg software and the JSSImporter processor. These will run on any Mac.

- First, install AutoPkg. It's just a package installer from GitHub.
- You also need the Xcode command line tools, because AutoPkg uses Git.
- [click] JSSImporter is not bundled in with AutoPkg, so next, you need to install that. It's also just a package installer available via GitHub.
- [click] You need to configure JSSImporter, because you have to give it access to Jamf Pro, and tell it how to connect.
- [click] You will need to make a user on your Jamf Pro server which has the necessary rights to interact with the Jamf Pro API.
- The JSSImporter wiki explains what to do.

ETH zürich

Setup AutoPkg + JSSImporter

autopkg.github.io/autopkg

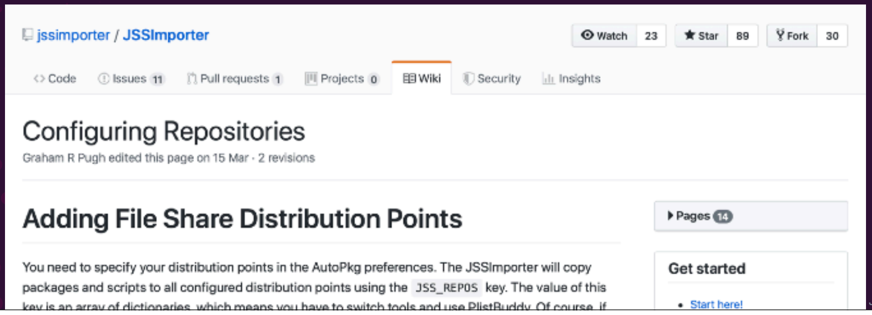
Setup AutoPkg + JSSImporter

autopkg.github.io/autopkg
github.com/jssimporter/JSSImporter

ETH zürich

Setup AutoPkg + JSSImporter

autopkg.github.io/autopkg
github.com/jssimporter/JSSImporter



jssimporter / JSSImporter

Watch 23 | Star 89 | Fork 30

Code | Issues 11 | Pull requests 1 | Projects 0 | Wiki | Security | Insights

Configuring Repositories

Graham R Pugh edited this page on 15 Mar · 2 revisions

Adding File Share Distribution Points

You need to specify your distribution points in the AutoPkg preferences. The JSSImporter will copy packages and scripts to all configured distribution points using the `JSS_REPOS` key. The value of this

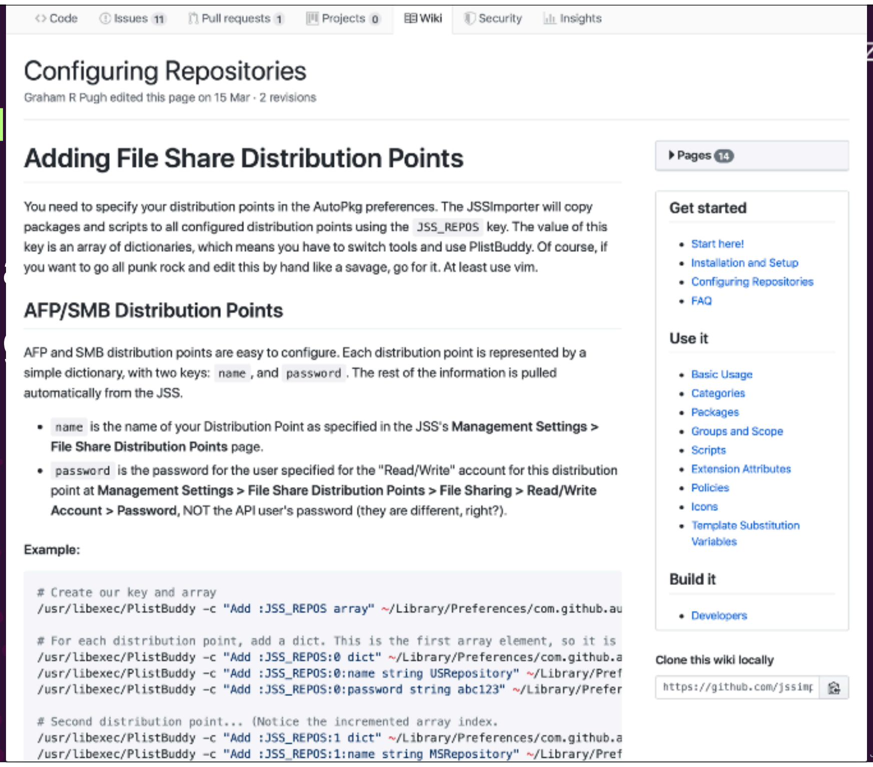
Pages 14

Get started

- Start here!

Setup Au

zürich



Configuring Repositories

Graham R Pugh edited this page on 15 Mar · 2 revisions

Adding File Share Distribution Points

You need to specify your distribution points in the AutoPkg preferences. The JSSImporter will copy packages and scripts to all configured distribution points using the `JSS_REPOS` key. The value of this key is an array of dictionaries, which means you have to switch tools and use PlistBuddy. Of course, if you want to go all punk rock and edit this by hand like a savage, go for it. At least use vim.

AFP/SMB Distribution Points

AFP and SMB distribution points are easy to configure. Each distribution point is represented by a simple dictionary, with two keys: `name` , and `password` . The rest of the information is pulled automatically from the JSS.

- `name` is the name of your Distribution Point as specified in the JSS's **Management Settings > File Share Distribution Points** page.
- `password` is the password for the user specified for the "Read/Write" account for this distribution point at **Management Settings > File Share Distribution Points > File Sharing > Read/Write Account > Password,** NOT the API user's password (they are different, right?).

Example:

```
# Create our key and array
/usr/libexec/PlistBuddy -c "Add :JSS_REPOS array" ~/Library/Preferences/com.github.au

# For each distribution point, add a dict. This is the first array element, so it is
/usr/libexec/PlistBuddy -c "Add :JSS_REPOS:0 dict" ~/Library/Preferences/com.github.a
/usr/libexec/PlistBuddy -c "Add :JSS_REPOS:0:name string USRepository" ~/Library/Pref
/usr/libexec/PlistBuddy -c "Add :JSS_REPOS:0:password string abc123" ~/Library/Prefer

# Second distribution point... (Notice the incremented array index.
/usr/libexec/PlistBuddy -c "Add :JSS_REPOS:1 dict" ~/Library/Preferences/com.github.a
/usr/libexec/PlistBuddy -c "Add :JSS_REPOS:1:name string MSRepository" ~/Library/Pref
```

Pages 14

Get started

- Start here!
- Installation and Setup
- Configuring Repositories
- FAQ

Use it

- Basic Usage
- Categories
- Packages
- Groups and Scope
- Scripts
- Extension Attributes
- Policies
- Icons
- Template Substitution Variables

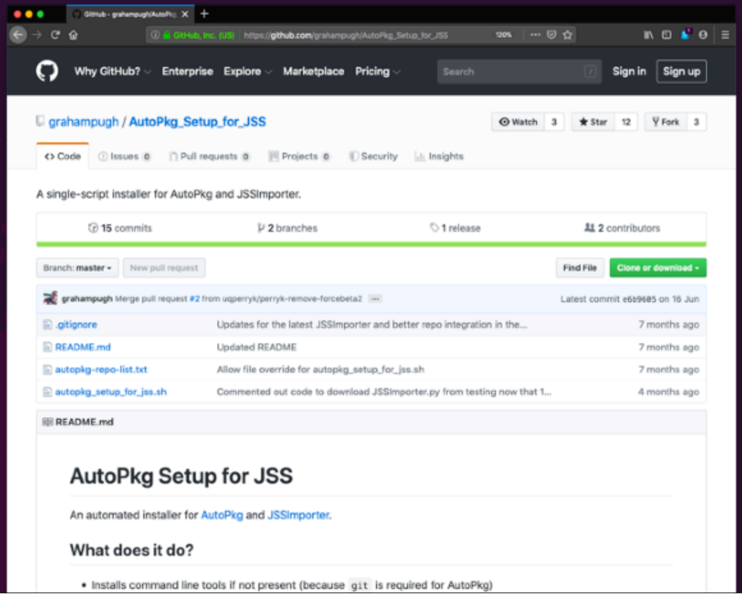
Build it

- Developers

Clone this wiki locally

`https://github.com/jssimp`

AutoPkg Setup for JSS



https://github.com/grahampugh/AutoPkg_Setup_for_JSS

grahampugh / AutoPkg_Setup_for_JSS

A single-script installer for AutoPkg and JSSImporter.

15 commits | 2 branches | 1 release | 2 contributors

grahampugh Merge pull request #2 from uqperryk/perryk-remove-forcebeta2 … Latest commit e6b9685 on 16 Jun

.gitignore	Updates for the latest JSSImporter and better repo integration in the…	7 months ago
README.md	Updated README	7 months ago
autopkg-repo-list.txt	Allow file override for autopkg_setup_for_jss.sh	7 months ago
autopkg_setup_for_jss.sh	Commented out code to download JSSImporter.py from testing now that 1…	4 months ago

README.md

AutoPkg Setup for JSS

An automated installer for AutoPkg and JSSImporter.

What does it do?

- Installs command line tools if not present (because `git` is required for AutoPkg)

- I also have a script which can do all that setup in one go called AutoPkg Setup for JSS, so go check that out.

Searching for Recipes

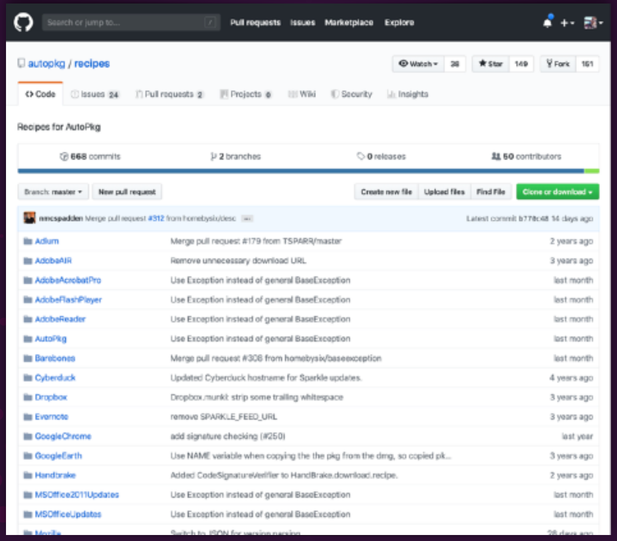
ETH zürich

© JAMF Software, LLC

Once AutoPkg is installed on your computer and JSSImporter is configured, we can start to obtain and run those recipes.

- [click] Lots of people in the MacAdmin community share recipes in GitHub repositories – thanks to all of you!
- Most are gathered into a single organisation which means that autopkg can find them.

Searching for Recipes



autopkg / recipes

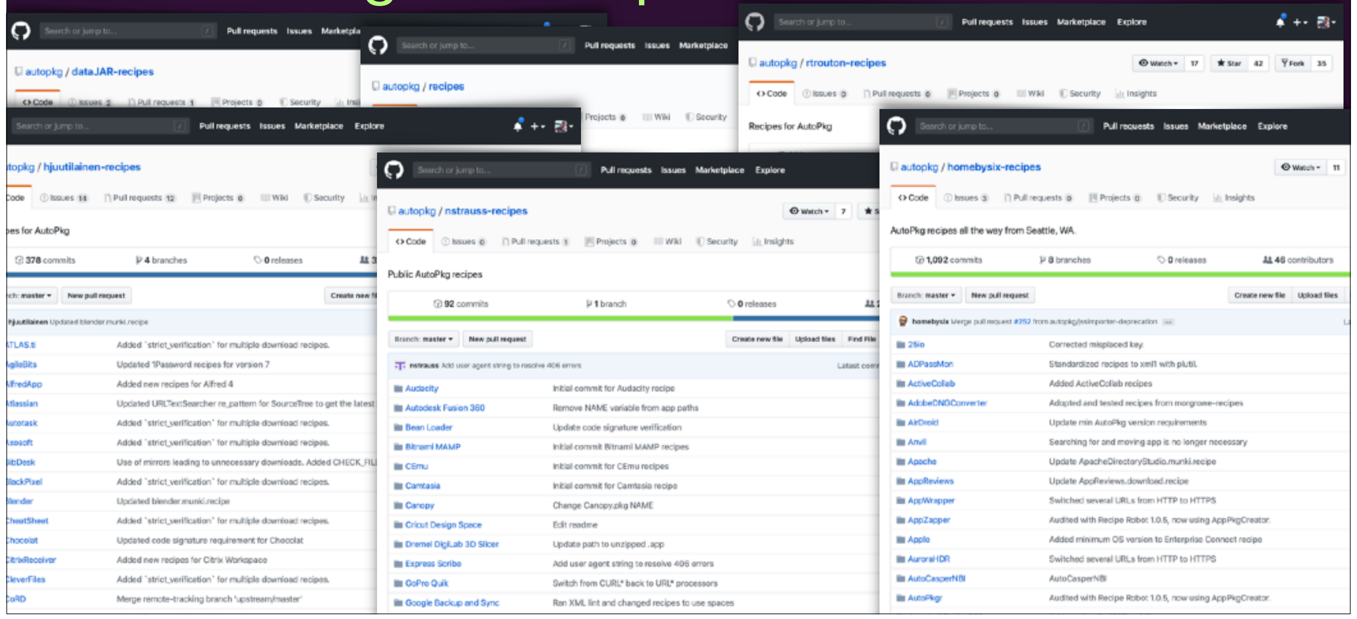
Recipes for AutoPkg

668 commits · 2 branches · 0 releases · 50 contributors

nmcspadden Merge pull request #312 from homebysix/desc … Latest commit b778c48 14 days ago

Adium	Merge pull request #179 from TSPARR/master	2 years ago
AdobeAIR	Remove unnecessary download URL	3 years ago
AdobeAcrobatPro	Use Exception instead of general BaseException	last month
AdobeFlashPlayer	Use Exception instead of general BaseException	last month
AdobeReader	Use Exception instead of general BaseException	last month
AutoPkg	Use Exception instead of general BaseException	last month
Barebones	Merge pull request #308 from homebysix/baseexception	last month
Cyberduck	Updated Cyberduck hostname for Sparkle updates.	4 years ago
Dropbox	Dropbox.munki: strip some trailing whitespace	3 years ago
Evernote	remove SPARKLE_FEED_URL	3 years ago
GoogleChrome	add signature checking (#250)	last year
GoogleEarth	Use NAME variable when copying the the pkg from the dmg, so copied pk…	3 years ago
Handbrake	Added CodeSignatureVerifier to HandBrake.download.recipe.	2 years ago
MSOffice2011Updates	Use Exception instead of general BaseException	last month
MSOfficeUpdates	Use Exception instead of general BaseException	last month
Mozilla	Switch to JSON for version parsing	28 days ago

Searching for Recipes



autopkg / dataJAR-recipes

Code | Issues 2 | Pull requests 1 | Projects 0 | Security

autopkg / recipes

autopkg / rtrouton-recipes

Watch 17 | Star 42 | Fork 35

Code | Issues 0 | Pull requests 0 | Projects 0 | Wiki | Security | Insights

Recipes for AutoPkg

topkg / hjuutilainen-recipes

Code | Issues 14 | Pull requests 12 | Projects 0 | Wiki | Security

es for AutoPkg

378 commits | 4 branches | 0 releases

hjuutilainen Updated blender.munki.recipe

TLAS.ti	Added `strict_verification` for multiple download recipes.
gileBits	Updated 1Password recipes for version 7
lfredApp	Added new recipes for Alfred 4
tlassian	Updated URLTextSearcher re_pattern for SourceTree to get the latest
utotask	Added `strict_verification` for multiple download recipes.
xosoft	Added `strict_verification` for multiple download recipes.
ibDesk	Use of mirrors leading to unnecessary downloads. Added CHECK_FIL
lackPixel	Added `strict_verification` for multiple download recipes.
lender	Updated blender.munki.recipe
heatSheet	Added `strict_verification` for multiple download recipes.
hocolat	Updated code signature requirement for Chocolat
itrixReceiver	Added new recipes for Citrix Workspace
leverFiles	Added `strict_verification` for multiple download recipes.
oRD	Merge remote-tracking branch 'upstream/master'

autopkg / nstrauss-recipes

Watch 7

Code | Issues 0 | Pull requests 1 | Projects 0 | Wiki | Security | Insights

Public AutoPkg recipes

92 commits | 1 branch | 0 releases

nstrauss Add user agent string to resolve 406 errors — Latest comm

Audacity	Initial commit for Audacity recipe
Autodesk Fusion 360	Remove NAME variable from app paths
Bean Loader	Update code signature verification
Bitnami MAMP	Initial commit Bitnami MAMP recipes
CEmu	Initial commit for CEmu recipes
Camtasia	Initial commit for Camtasia recipe
Canopy	Change Canopy.pkg NAME
Cricut Design Space	Edit readme
Dremel DigiLab 3D Slicer	Update path to unzipped .app
Express Scribe	Add user agent string to resolve 406 errors
GoPro Quik	Switch from CURL* back to URL* processors
Google Backup and Sync	Ran XML lint and changed recipes to use spaces

autopkg / homebysix-recipes

Watch 11

Code | Issues 3 | Pull requests 0 | Projects 0 | Security | Insights

AutoPkg recipes all the way from Seattle, WA.

1,092 commits | 8 branches | 0 releases | 48 contributors

homebysix Merge pull request #252 from autopkg/plsimporter-deprecation

2Do	Corrected misplaced key.
ADPassMon	Standardized recipes to xml1 with plutil.
ActiveCollab	Added ActiveCollab recipes
AdobeDNGConverter	Adopted and tested recipes from mosgrove-recipes
AirDroid	Update min AutoPkg version requirements
Anvil	Searching for and moving app is no longer necessary
Apache	Update ApacheDirectoryStudio.munki.recipe
AppReviews	Update AppReviews.download.recipe
AppWrapper	Switched several URLs from HTTP to HTTPS
AppZapper	Audited with Recipe Robot 1.0.5, now using AppPkgCreator.
Apple	Added minimum OS version to Enterprise Connect recipe
AuroraHDR	Switched several URLs from HTTP to HTTPS
AutoCasperNBI	AutoCasperNBI
AutoPkgr	Audited with Recipe Robot 1.0.5, now using AppPkgCreator.

Searching for Recipes

Searching for Recipes

ETH zürich

© JAMF Software, LLC

- AutoPkg is an easy-to-use command line tool.
- To search for a recipe, run autopkg with the verb 'search' and the name of an app.
- [click] I get a list of possible recipes - I want the jss recipe since I use Jamf Pro.
- [click] One exists, so we can use that.
- [click] It's in the repo named jss-recipes, so let's download the contents of that to our computer with the verb 'repo-add'.

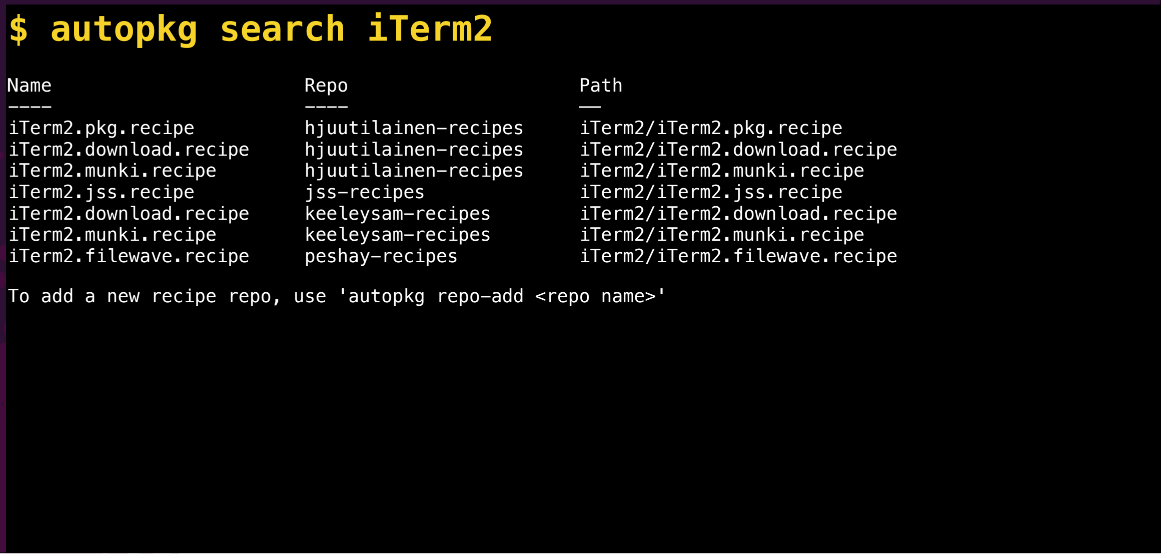
Searching for Recipes

```
$
```

Searching for Recipes

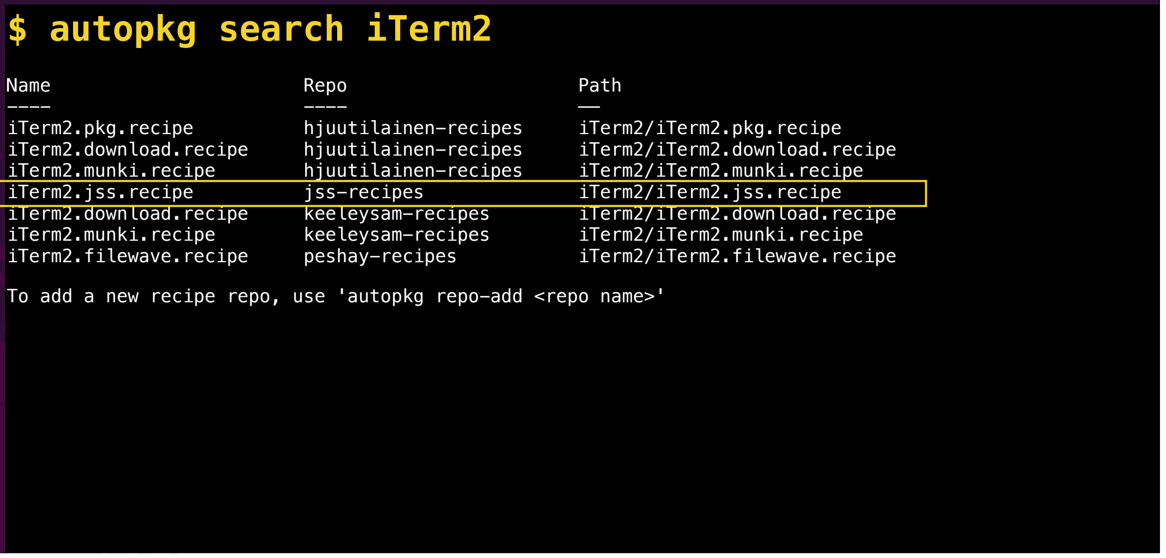
```
$ autopkg search iTerm2
```

Searching for Recipes



```
$ autopkg search iTerm2

Name                       Repo                     Path
----                       ----                     --
iTerm2.pkg.recipe          hjuutilainen-recipes     iTerm2/iTerm2.pkg.recipe
iTerm2.download.recipe     hjuutilainen-recipes     iTerm2/iTerm2.download.recipe
iTerm2.munki.recipe        hjuutilainen-recipes     iTerm2/iTerm2.munki.recipe
iTerm2.jss.recipe          jss-recipes              iTerm2/iTerm2.jss.recipe
iTerm2.download.recipe     keeleysam-recipes        iTerm2/iTerm2.download.recipe
iTerm2.munki.recipe        keeleysam-recipes        iTerm2/iTerm2.munki.recipe
iTerm2.filewave.recipe     peshay-recipes           iTerm2/iTerm2.filewave.recipe

To add a new recipe repo, use 'autopkg repo-add <repo name>'
```

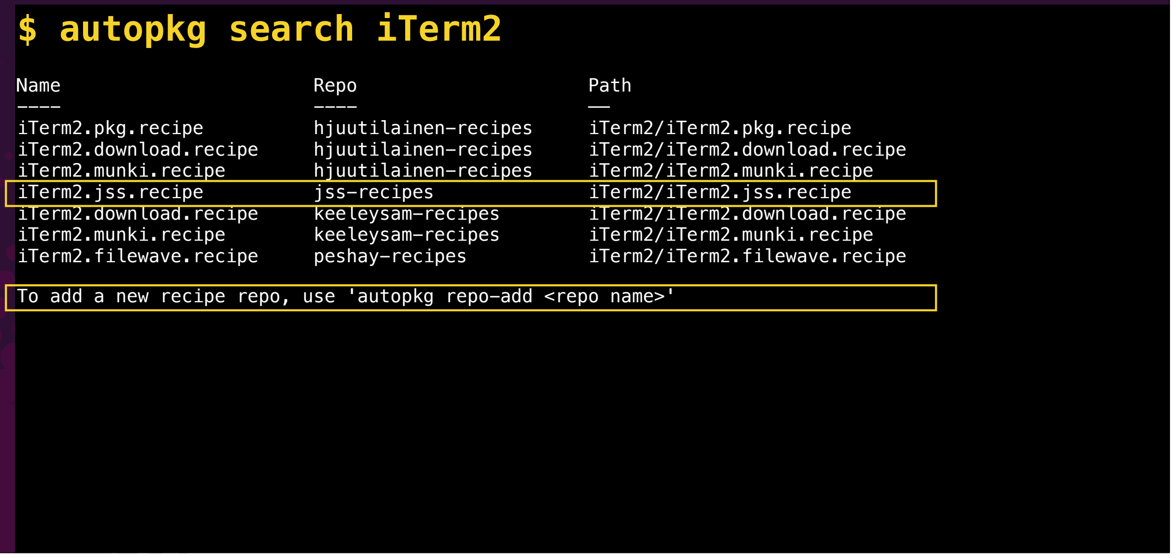
Searching for Recipes



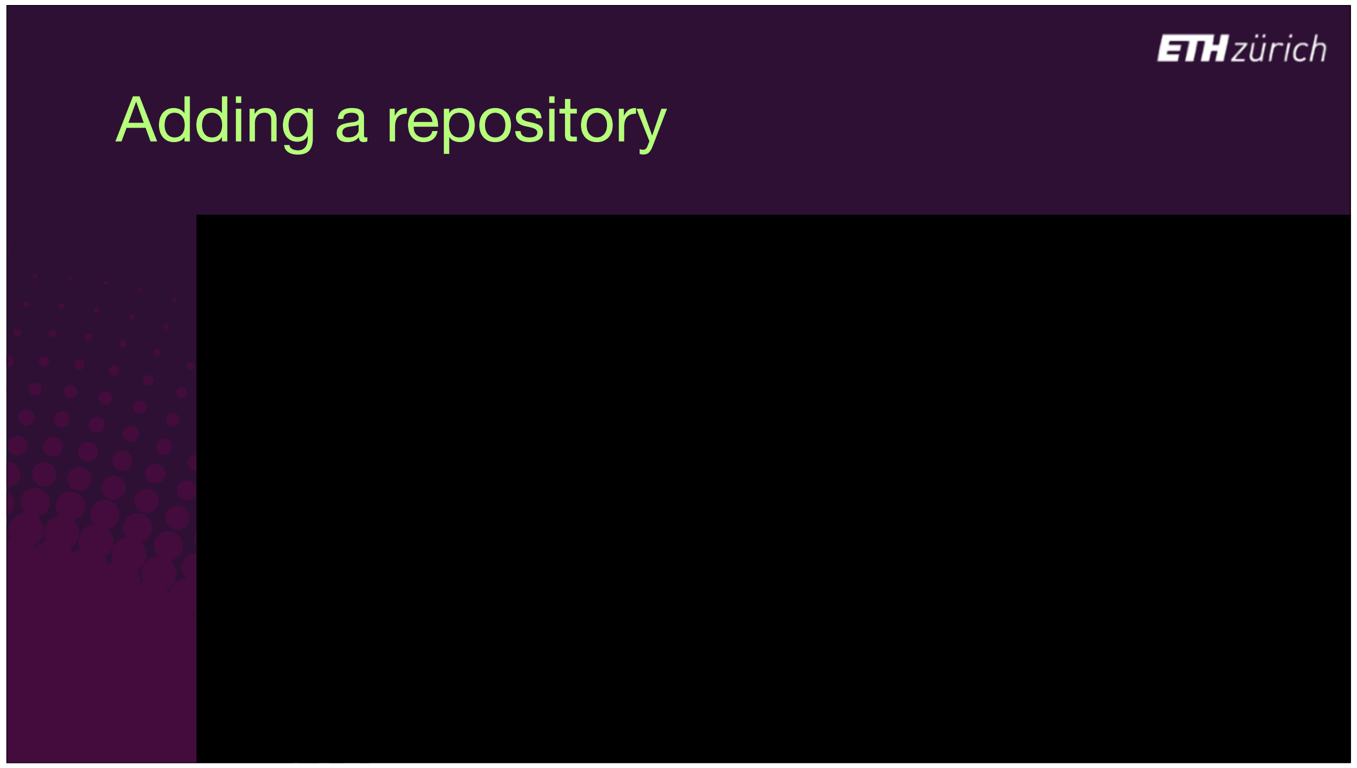
```
$ autopkg search iTerm2

Name                       Repo                      Path
----                       ----                      --
iTerm2.pkg.recipe          hjuutilainen-recipes      iTerm2/iTerm2.pkg.recipe
iTerm2.download.recipe     hjuutilainen-recipes      iTerm2/iTerm2.download.recipe
iTerm2.munki.recipe        hjuutilainen-recipes      iTerm2/iTerm2.munki.recipe
iTerm2.jss.recipe          jss-recipes               iTerm2/iTerm2.jss.recipe
iTerm2.download.recipe     keeleysam-recipes         iTerm2/iTerm2.download.recipe
iTerm2.munki.recipe        keeleysam-recipes         iTerm2/iTerm2.munki.recipe
iTerm2.filewave.recipe     peshay-recipes            iTerm2/iTerm2.filewave.recipe

To add a new recipe repo, use 'autopkg repo-add <repo name>'
```

ETH zürich

Searching for Recipes



```
$ autopkg search iTerm2

Name                       Repo                      Path
----                       ----                      --
iTerm2.pkg.recipe          hjuutilainen-recipes      iTerm2/iTerm2.pkg.recipe
iTerm2.download.recipe     hjuutilainen-recipes      iTerm2/iTerm2.download.recipe
iTerm2.munki.recipe        hjuutilainen-recipes      iTerm2/iTerm2.munki.recipe
iTerm2.jss.recipe          jss-recipes               iTerm2/iTerm2.jss.recipe
iTerm2.download.recipe     keeleysam-recipes         iTerm2/iTerm2.download.recipe
iTerm2.munki.recipe        keeleysam-recipes         iTerm2/iTerm2.munki.recipe
iTerm2.filewave.recipe     peshay-recipes            iTerm2/iTerm2.filewave.recipe

To add a new recipe repo, use 'autopkg repo-add <repo name>'
```

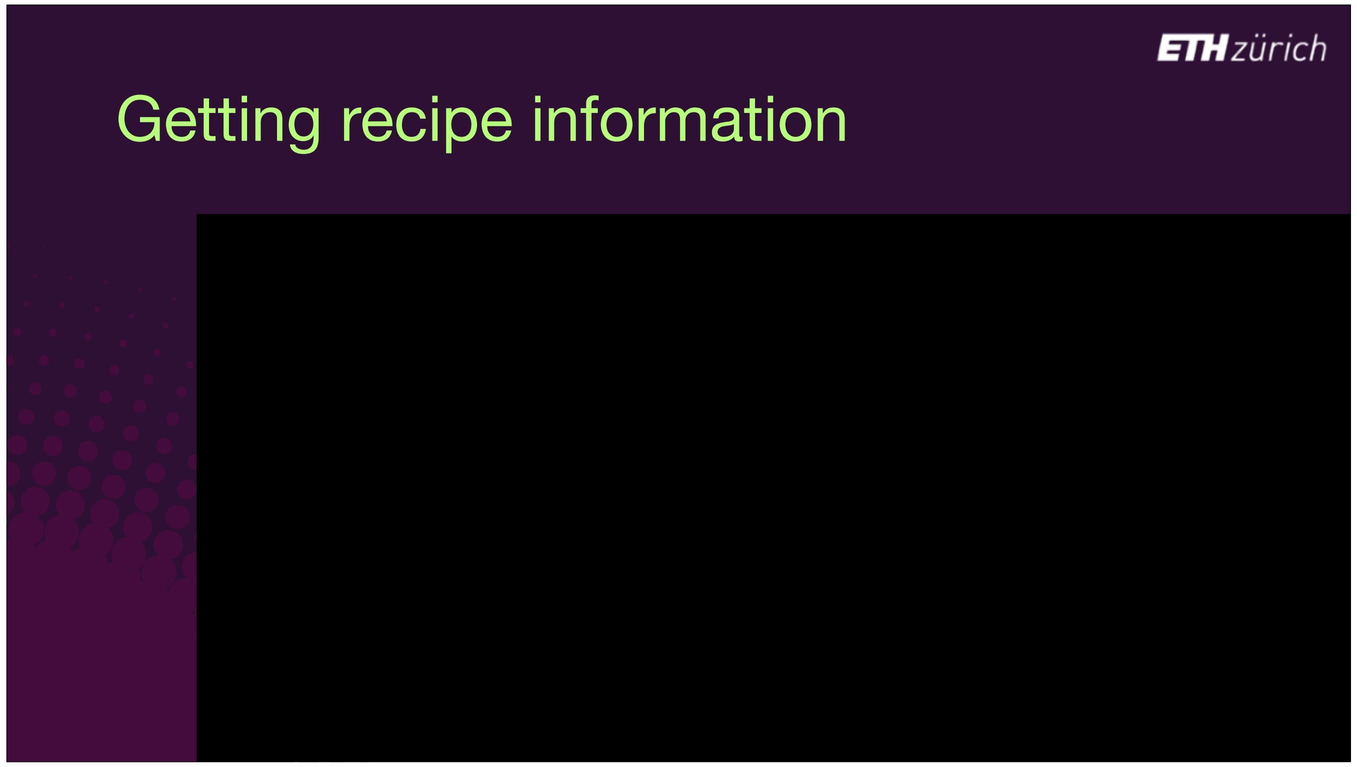


Adding a repository

- The repo is cloned to our computer using git.

Adding a repository

```
$ autopkg repo-add jss-recipes

Attempting git clone...

Adding /Users/g/Library/AutoPkg/RecipeRepos/com.github.autopkg.jss-recipes to RECIPE_SEARCH_DIRS…

Updated search path:
  '.'
  '~/Library/AutoPkg/Recipes'
  '/Library/AutoPkg/Recipes'
  '/Users/g/Library/AutoPkg/RecipeRepos/com.github.autopkg.jss-recipes'
```



Getting recipe information

We need to make sure we also have the repos of the parent recipes on our computer before we can actually run the recipe.

- To check this, we can run the verb 'info'.
- [click] It looks inside the recipe for the identifier of the parent, and then checks our local search path to see if we have that recipe.
- If we don't, so it offers to search GitHub for it.
- [click] It's in the “hjuutilainen-recipes” repo, so let's add that.

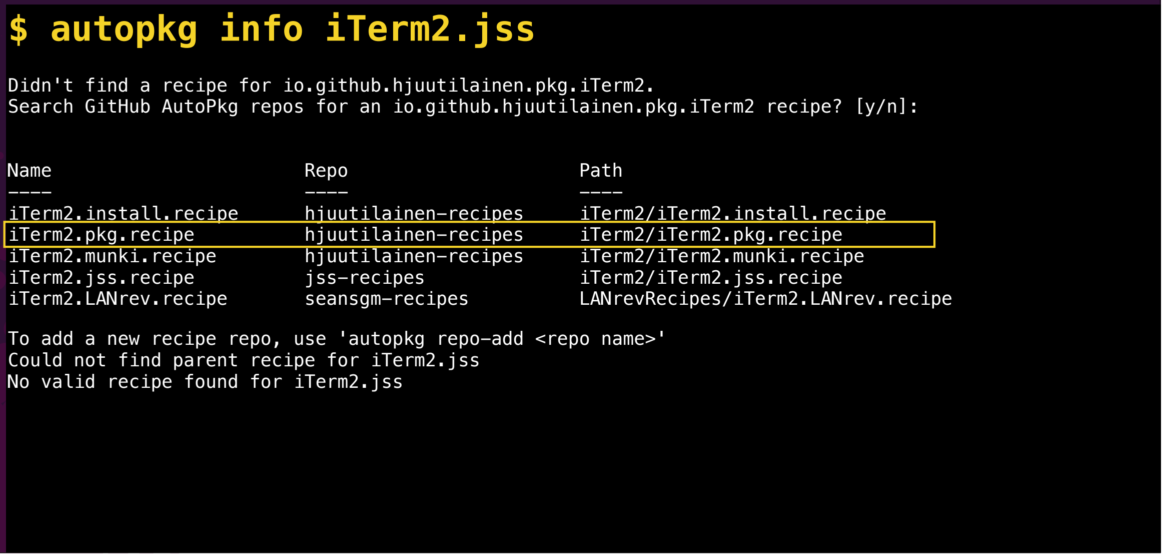
Getting recipe information

```
$ autopkg info iTerm2.jss
```

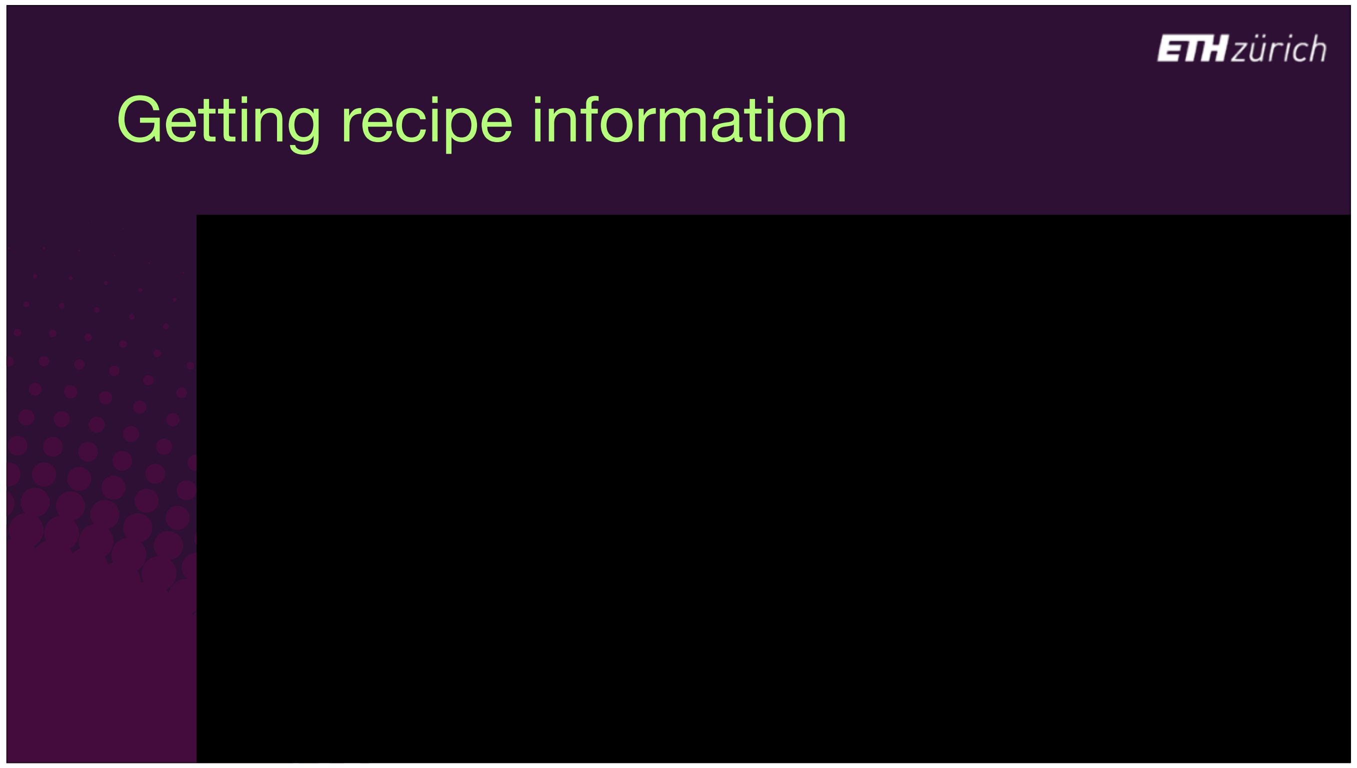
Getting recipe information

```
$ autopkg info iTerm2.jss

Didn't find a recipe for io.github.hjuutilainen.pkg.iTerm2.
Search GitHub AutoPkg repos for an io.github.hjuutilainen.pkg.iTerm2 recipe? [y/n]:
```

Getting recipe information



```
$ autopkg info iTerm2.jss

Didn't find a recipe for io.github.hjuutilainen.pkg.iTerm2.
Search GitHub AutoPkg repos for an io.github.hjuutilainen.pkg.iTerm2 recipe? [y/n]:


Name                      Repo                     Path
----                      ----                     ----
iTerm2.install.recipe     hjuutilainen-recipes     iTerm2/iTerm2.install.recipe
iTerm2.pkg.recipe         hjuutilainen-recipes     iTerm2/iTerm2.pkg.recipe
iTerm2.munki.recipe       hjuutilainen-recipes     iTerm2/iTerm2.munki.recipe
iTerm2.jss.recipe         jss-recipes              iTerm2/iTerm2.jss.recipe
iTerm2.LANrev.recipe      seansgm-recipes          LANrevRecipes/iTerm2.LANrev.recipe

To add a new recipe repo, use 'autopkg repo-add <repo name>'
Could not find parent recipe for iTerm2.jss
No valid recipe found for iTerm2.jss
```

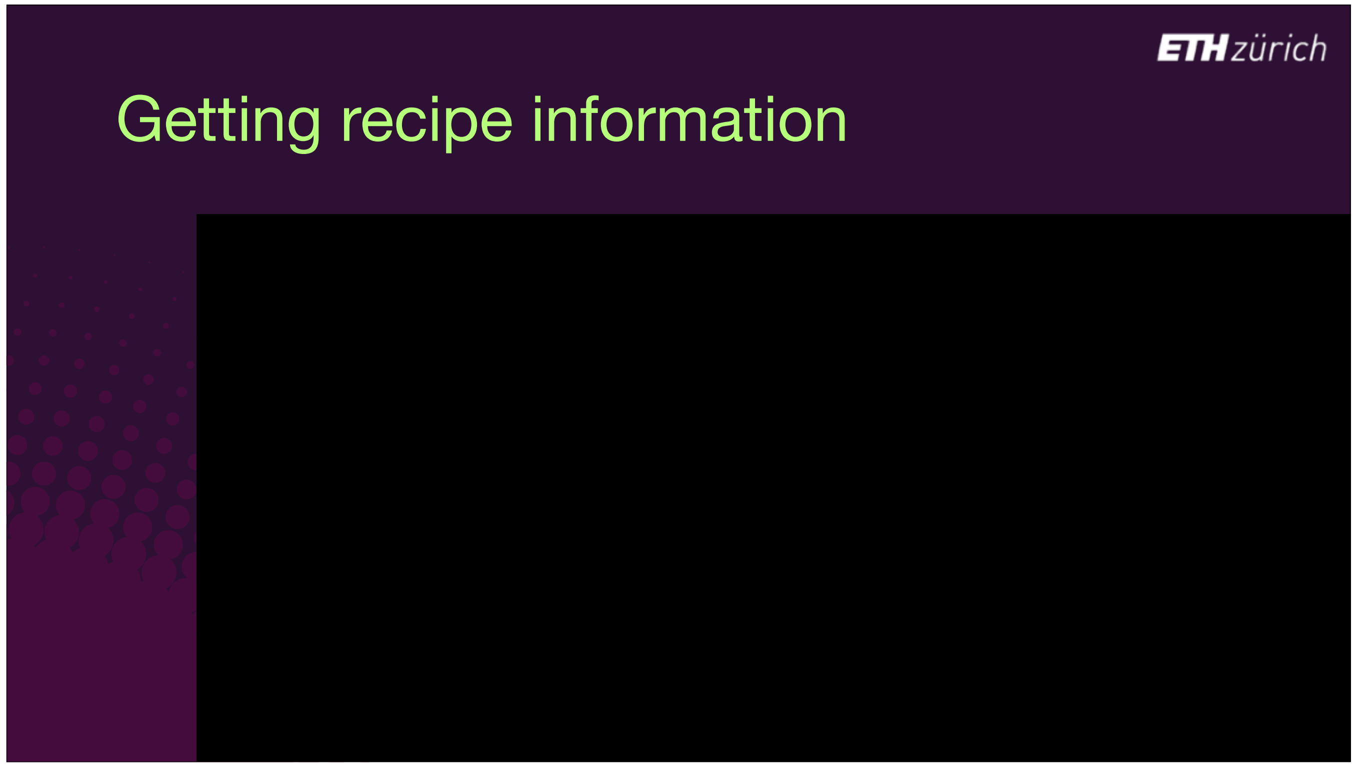


Getting recipe information

So we add that repo,
And the downloaded clone is added to the search path below.

Getting recipe information

```
$ autopkg repo-add hjuutilainen-recipes

Attempting git clone...

Adding /Users/g/Library/AutoPkg/RecipeRepos/com.github.autopkg.hjuutilainen-recipes to
RECIPE_SEARCH_DIRS…

Updated search path:
  '.'
  '~/Library/AutoPkg/Recipes'
  '/Library/AutoPkg/Recipes'
  '/Users/g/Library/AutoPkg/RecipeRepos/com.github.autopkg.jss-recipes'
  '/Users/g/Library/AutoPkg/RecipeRepos/com.github.autopkg.hjuutilainen-recipes'
```

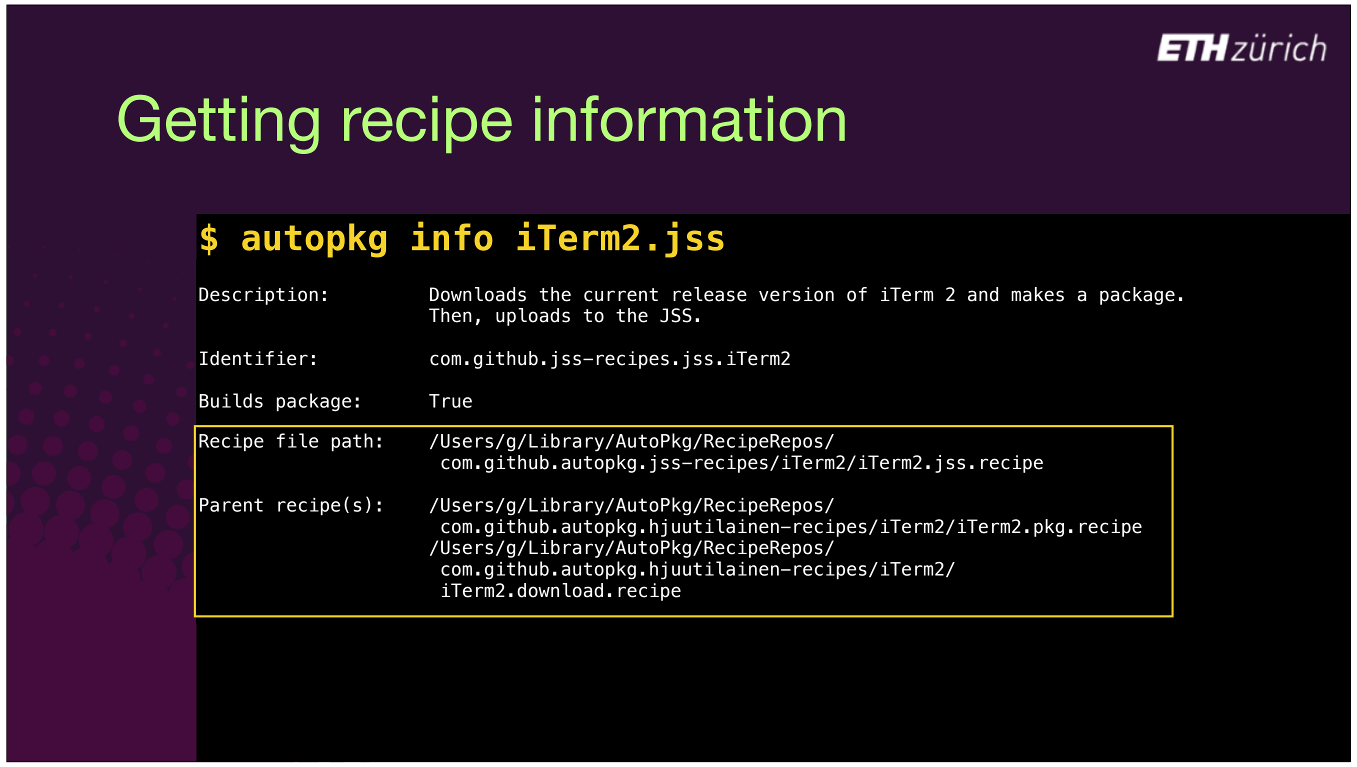


Getting recipe information

Once we have all the repos we need locally, the “info” verb gives you a bunch of info about the recipe, including the paths to all the recipe files.

Getting recipe information

```
$ autopkg info iTerm2.jss

Description:         Downloads the current release version of iTerm 2 and makes a package.
                     Then, uploads to the JSS.

Identifier:          com.github.jss-recipes.jss.iTerm2

Builds package:      True

Recipe file path:    /Users/g/Library/AutoPkg/RecipeRepos/
                      com.github.autopkg.jss-recipes/iTerm2/iTerm2.jss.recipe

Parent recipe(s):    /Users/g/Library/AutoPkg/RecipeRepos/
                      com.github.autopkg.hjuutilainen-recipes/iTerm2/iTerm2.pkg.recipe
                     /Users/g/Library/AutoPkg/RecipeRepos/
                      com.github.autopkg.hjuutilainen-recipes/iTerm2/
                      iTerm2.download.recipe
```



Getting recipe information

```
$ autopkg info iTerm2.jss

Description:         Downloads the current release version of iTerm 2 and makes a package.
                     Then, uploads to the JSS.

Identifier:          com.github.jss-recipes.jss.iTerm2

Builds package:      True

Recipe file path:    /Users/g/Library/AutoPkg/RecipeRepos/
                      com.github.autopkg.jss-recipes/iTerm2/iTerm2.jss.recipe

Parent recipe(s):    /Users/g/Library/AutoPkg/RecipeRepos/
                      com.github.autopkg.hjuutilainen-recipes/iTerm2/iTerm2.pkg.recipe
                     /Users/g/Library/AutoPkg/RecipeRepos/
                      com.github.autopkg.hjuutilainen-recipes/iTerm2/
                      iTerm2.download.recipe
```

- What you need to do now is go and look at all those files and make sure you understand what they are doing, and that you trust them.
- AutoPkg is really powerful, because it builds packages that you could have no idea about, which you then install on all your clients using Jamf, which of course runs as root on those clients.
- That's obviously dangerous, so you need to make sure you know what's in those packages to protect your users.
- Fortunately, AutoPkg has a built in safety feature to make sure you do go check those files.

ETH zürich

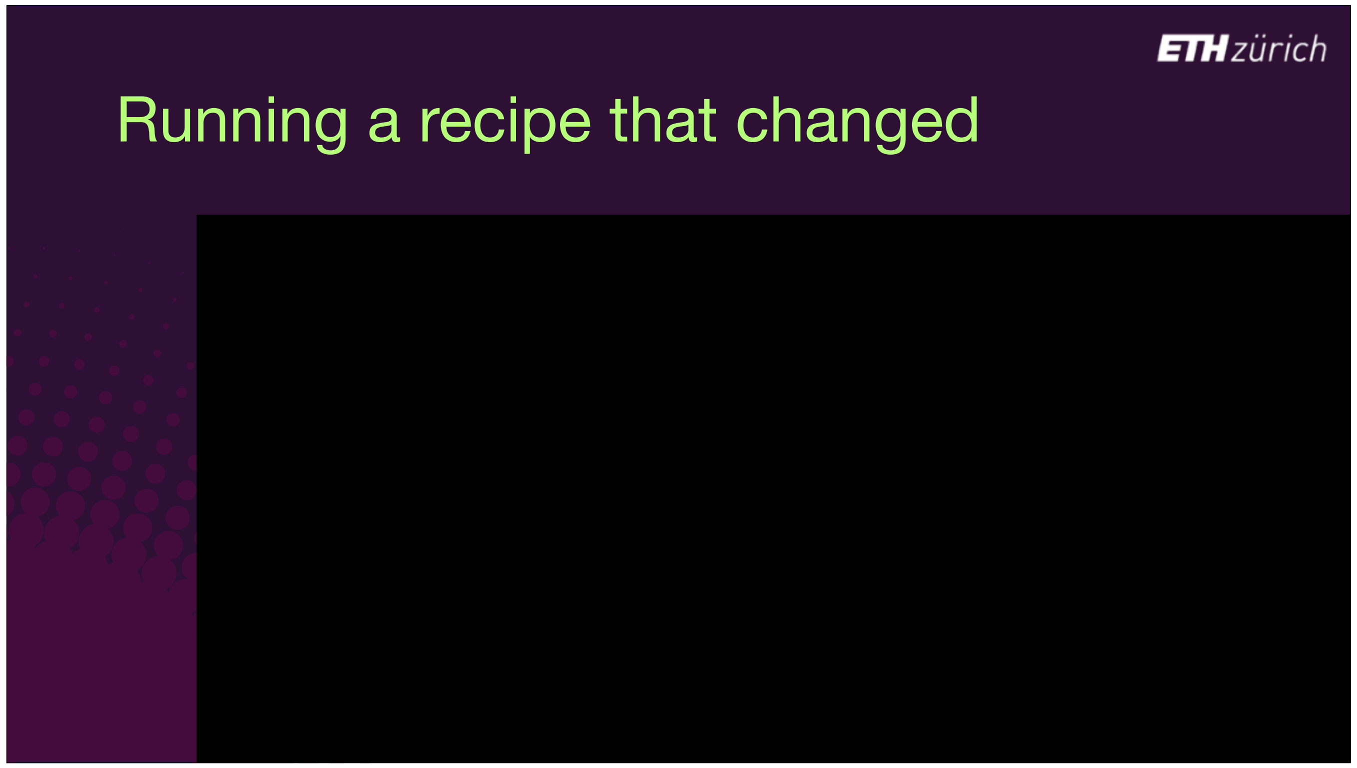
Getting recipe information

- This security feature is not enabled by default, but I strongly urge you to set it.
- To set it as the default, run this defaults write command.

ETH zürich

Getting recipe information

```
$ defaults write com.github.autopkg
  FAIL_RECIPES_WITHOUT_TRUST_INFO -bool true
```



Running a recipe that changed

We run a recipe with the verb 'run'.
If you run the recipe without first telling AutoPkg that you trust it,

- [click] the recipe fails, saying "No trust information present".
- This is your cue to go verify that the files are OK.

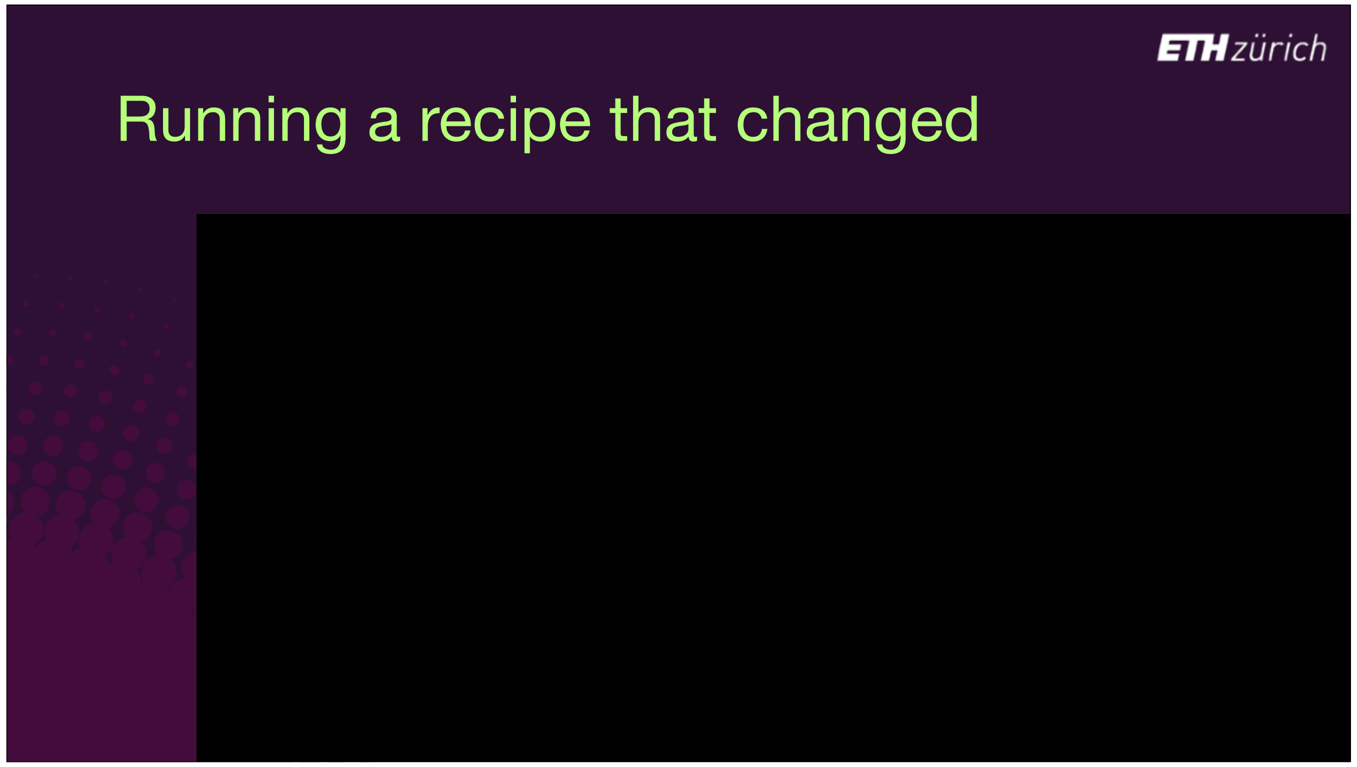
Running a recipe that changed

```
$ autopkg run iTerm2.jss
```

ETH zürich

Running a recipe that changed

```
$ autopkg run iTerm2.jss

Failed local trust verification.

The following recipes failed:
    iTerm2.jss
        No trust information present.

Nothing downloaded, packaged or imported.
```

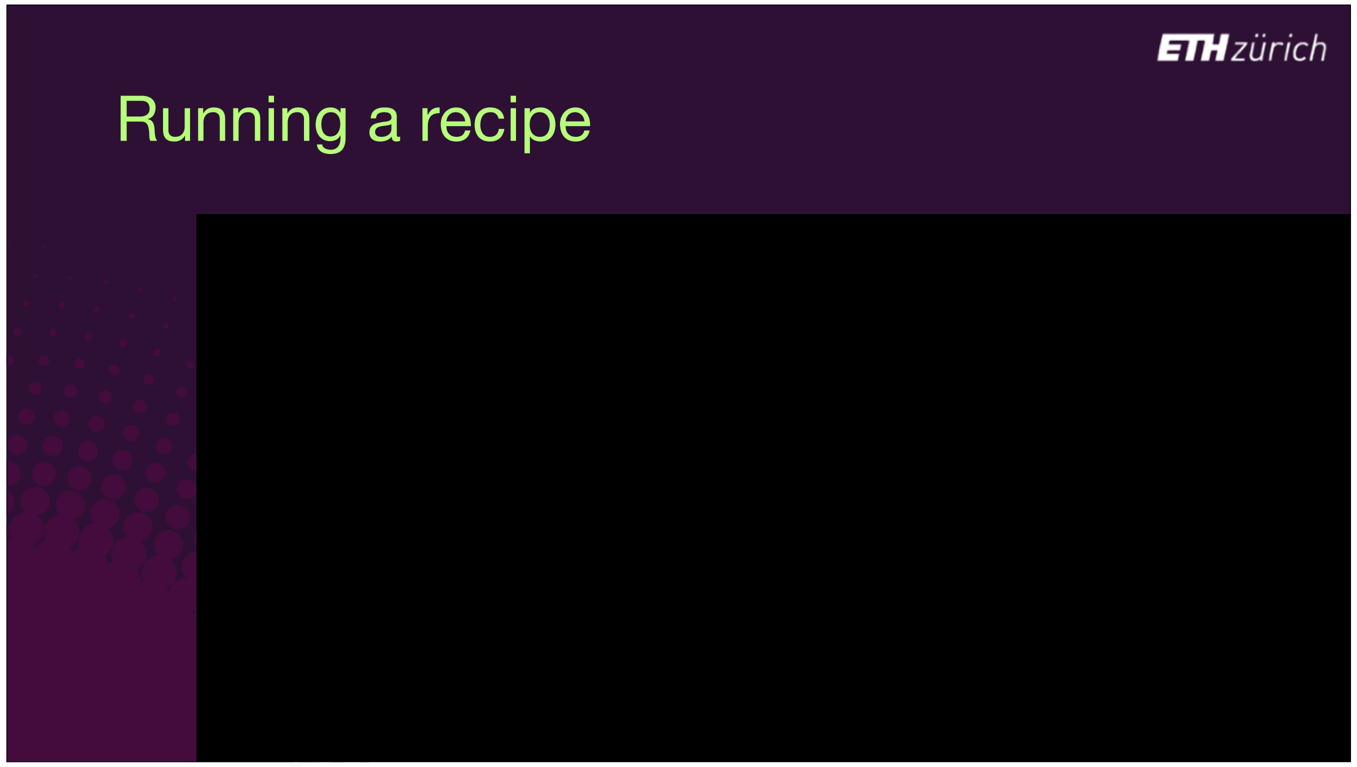

ETH zürich

Running a recipe that changed

- Once you are happy with the recipe contents, run the "make-override" verb.
- This makes a new local recipe file called a recipe override.
- This is a special recipe which has the main JSS recipe as its parent.
- It includes trust information about each of the parent recipes - basically, the hash of each file.

Running a recipe that changed

```
$ autopkg make-override iTerm2.jss

Override file saved to /Users/g/Library/AutoPkg/RecipeOverrides/iTerm2.jss.recipe
```

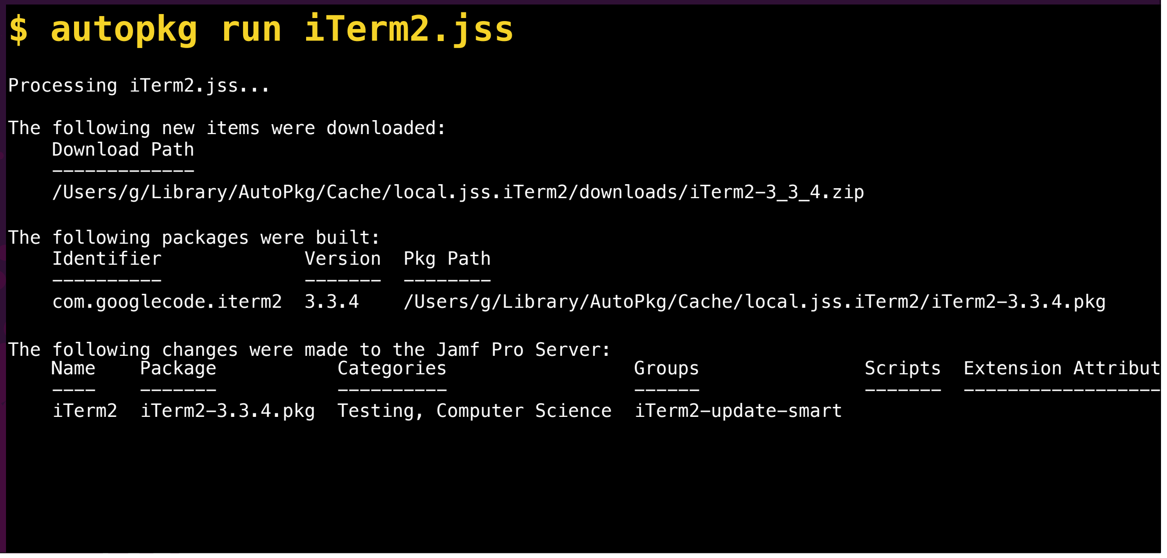


Running a recipe

ETH zürich

Now when we run the recipe, it checks the trust info is correct in the override, and proceeds…
1. A new item is downloaded to a cache folder – in this case a ZIP file.
2. A package is built and the version is identified.
3. And a bunch of changes were made to the Jamf Pro server.
    - [click] new categories, a group, the policy object, icon etc.

Running a recipe

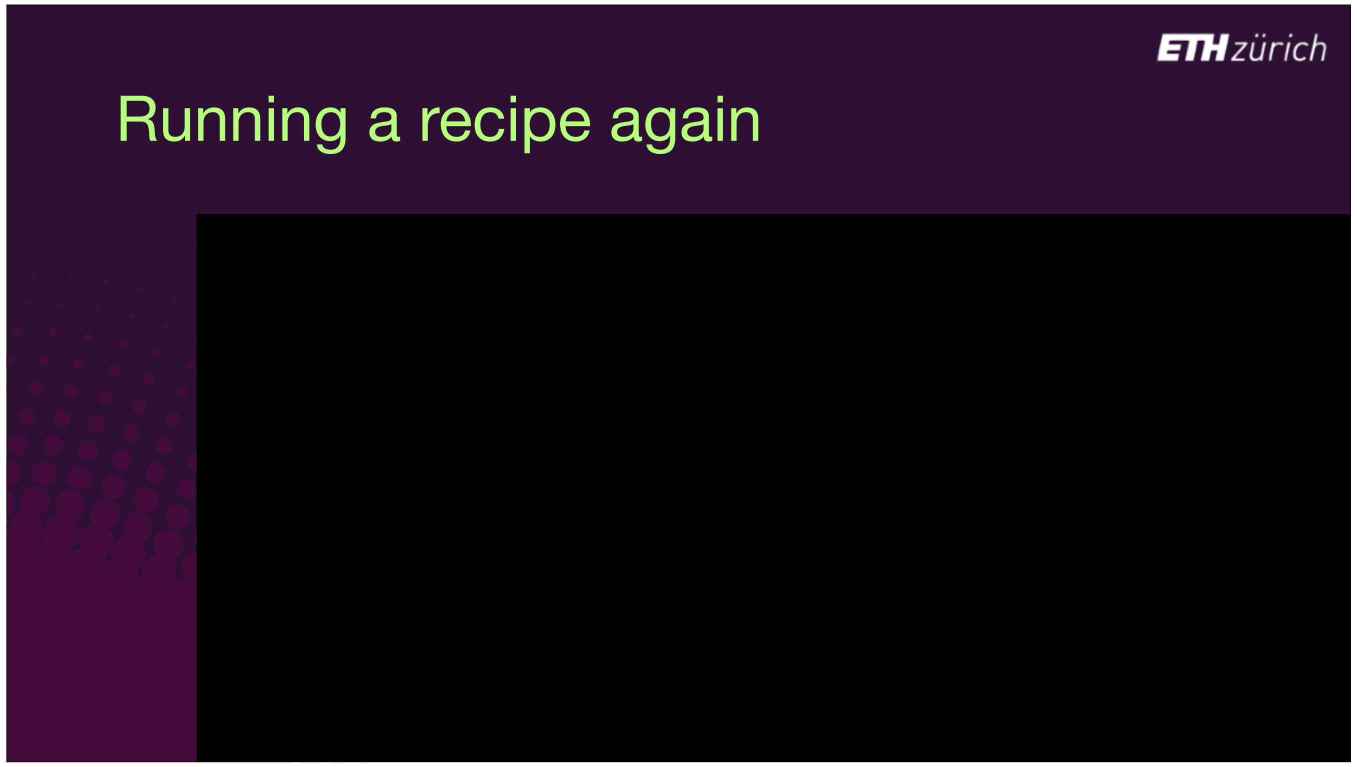


```
$ autopkg run iTerm2.jss

Processing iTerm2.jss...

The following new items were downloaded:
    Download Path
    -------------
    /Users/g/Library/AutoPkg/Cache/local.jss.iTerm2/downloads/iTerm2-3_3_4.zip

The following packages were built:
    Identifier            Version  Pkg Path
    ----------            -------  --------
    com.googlecode.iterm2  3.3.4    /Users/g/Library/AutoPkg/Cache/local.jss.iTerm2/iTerm2-3.3.4.pkg

The following changes were made to the Jamf Pro Server:
    Name    Package           Categories                Groups               Scripts  Extension Attribut
    ----    -------           ----------                ------               -------  ------------------
    iTerm2  iTerm2-3.3.4.pkg  Testing, Computer Science  iTerm2-update-smart
```

ETH zürich

Running a recipe

```
$ autopkg run iTerm2.jss

Processing iTerm2.jss...

The following new items were downloaded:
    Download Path
    -------------
    /Users/g/Library/AutoPkg/Cache/local.jss.iTerm2/downloads/iTerm2-3_3_4.zip

The following packages were built:
    Identifier             Version  Pkg Path
    ----------             -------  --------
    com.googlecode.iterm2  3.3.4    /Users/g/Library/AutoPkg/Cache/local.jss.iTerm2/iTerm2-3.3.4.pkg

Server:
         Groups               Scripts  Extension Attributes  Policy                 Icon        Version  Package Uploaded
         ------               -------  --------------------  ------                 ----        -------  ----------------
Science  iTerm2-update-smart                                 Install Latest iTerm2  iTerm2.png  3.3.4    True
```

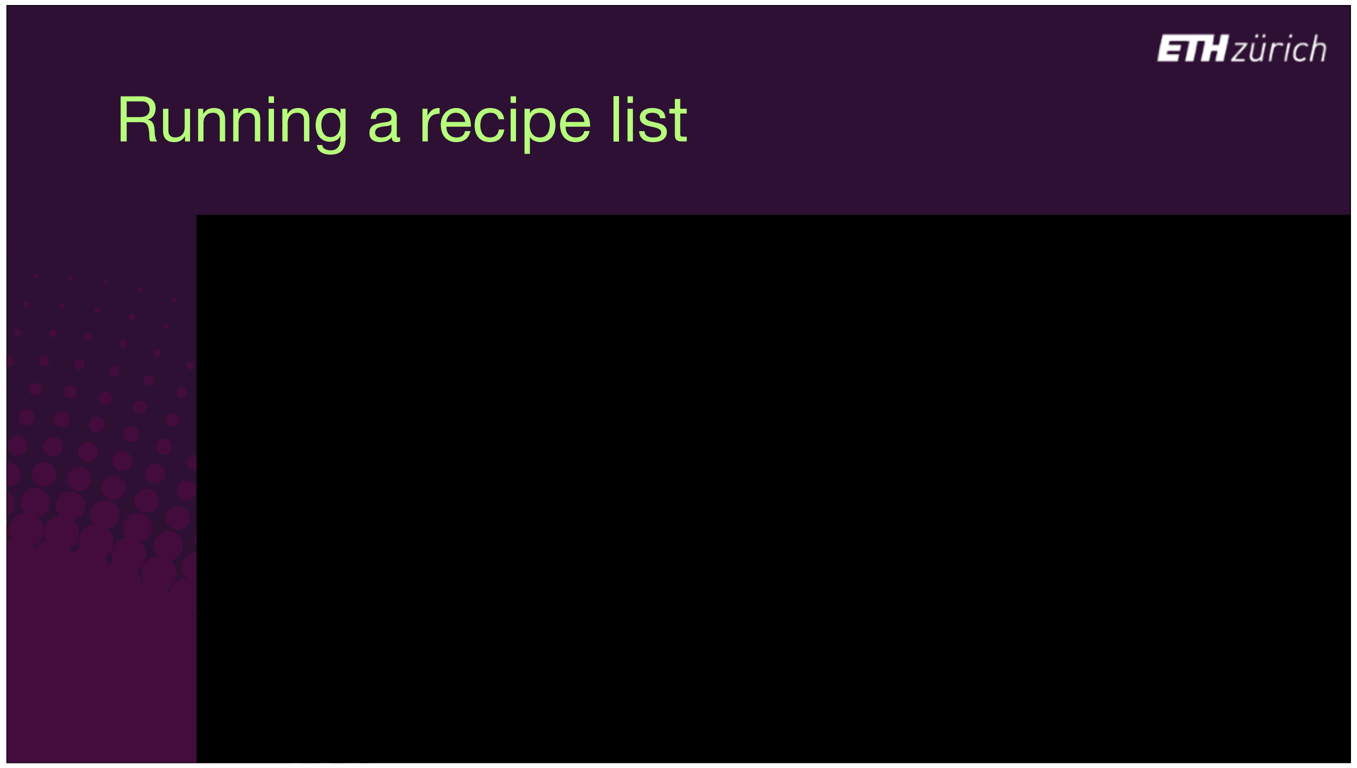


Running a recipe again

If we run the recipe again, and the source files haven't changed, nothing happens.
This means you can safely run the recipe on a schedule, like every night.

Running a recipe again

```
$ autopkg run iTerm2.jss

Processing iTerm2.jss...

Nothing downloaded, packaged or imported.
```



Running a recipe list

It also means you can run all your recipes at once using a recipe list file.
[click] Create a text file containing all your recipes, and run with the 'recipe-list' argument.

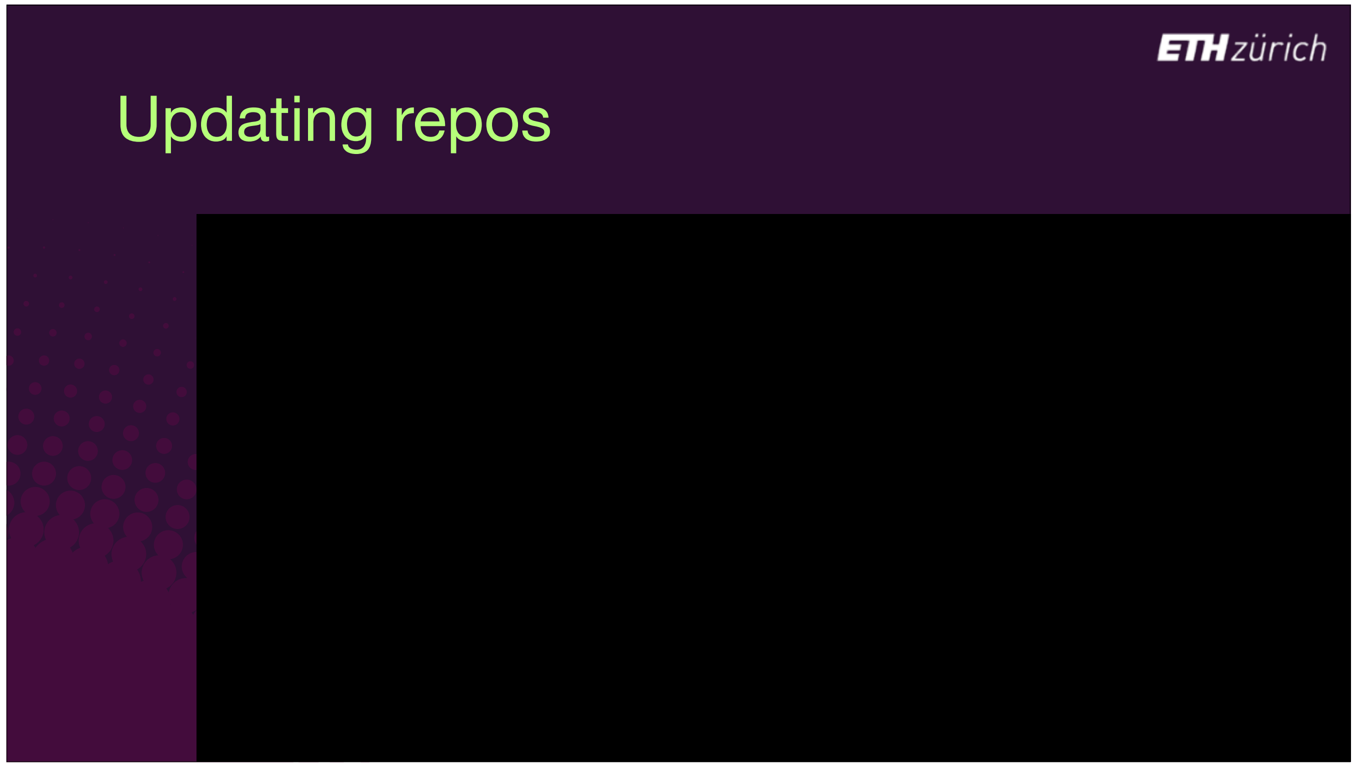
ETH zürich

Running a recipe list

```
$ autopkg run --recipe-list JSS_Recipes.txt
```

Running a recipe list

```
$ autopkg run --recipe-list JSS_Recipes.txt


Papers.jss
Carbon Copy Cloner.jss
VMware Horizon Client.jss
Adobe Acrobat Reader DC.jss
BBEdit.jss
FileZilla.jss
GraphicConverter 10.jss
R.jss
RStudio.jss
Adobe Flash Player.jss
Google Chrome.jss
Skype.jss
Microsoft Remote Desktop.jss
LibreOffice.jss
ISL Light Client.jss
GIMP.jss
DataWarrior.jss
XQuartz.jss
PopChar.jss
Quicksilver.jss
PyMOL.jss
```



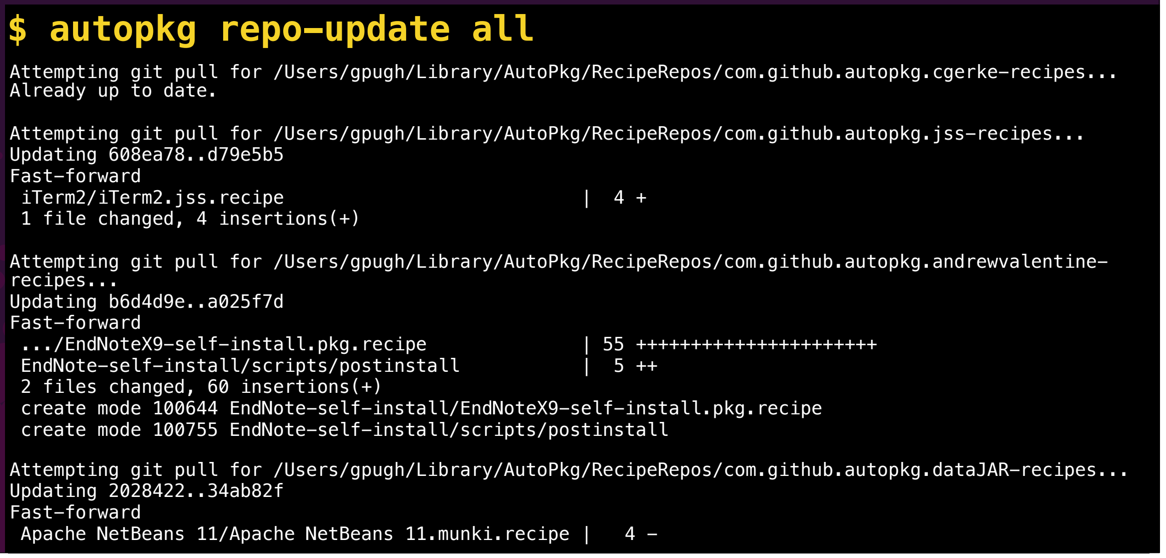
Updating repos

Recipes can get updated over time, for instance because the source URL provided by the vendor might change, so after a while you should synchronise your repos, using "repo-update all"

- Any remote changes are downloaded.
- [click] You see here that our iTerm2 jss recipe has changed.

ETH zürich

Updating repos

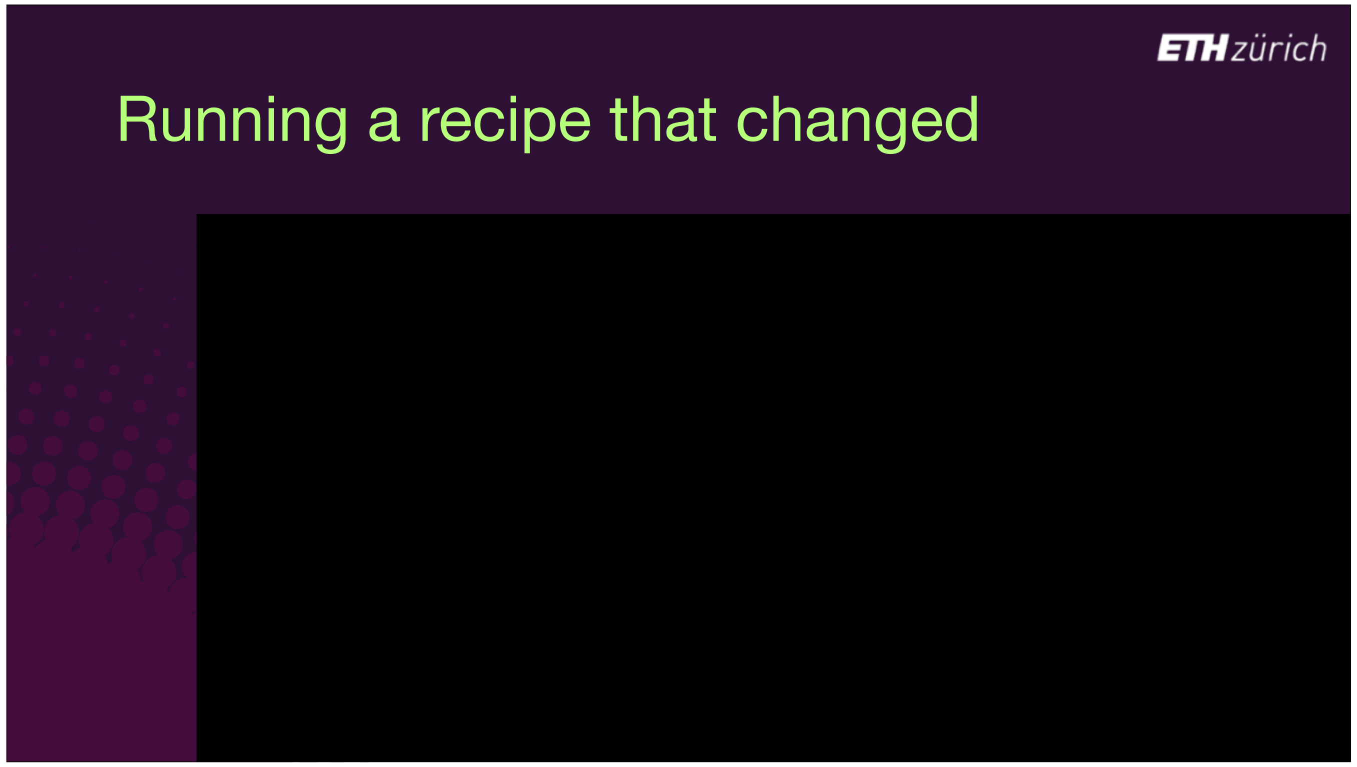


```
$ autopkg repo-update all
Attempting git pull for /Users/gpugh/Library/AutoPkg/RecipeRepos/com.github.autopkg.cgerke-recipes...
Already up to date.

Attempting git pull for /Users/gpugh/Library/AutoPkg/RecipeRepos/com.github.autopkg.jss-recipes...
Updating 608ea78..d79e5b5
Fast-forward
 iTerm2/iTerm2.jss.recipe                                  |  4 +
 1 file changed, 4 insertions(+)

Attempting git pull for /Users/gpugh/Library/AutoPkg/RecipeRepos/com.github.autopkg.andrewvalentine-
recipes...
Updating b6d4d9e..a025f7d
Fast-forward
 .../EndNoteX9-self-install.pkg.recipe              | 55 ++++++++++++++++++++++
 EndNote-self-install/scripts/postinstall           |  5 ++
 2 files changed, 60 insertions(+)
 create mode 100644 EndNote-self-install/EndNoteX9-self-install.pkg.recipe
 create mode 100755 EndNote-self-install/scripts/postinstall

Attempting git pull for /Users/gpugh/Library/AutoPkg/RecipeRepos/com.github.autopkg.dataJAR-recipes...
Updating 2028422..34ab82f
Fast-forward
 Apache NetBeans 11/Apache NetBeans 11.munki.recipe |   4 -
```

ETH zürich

Updating repos

```
$ autopkg repo-update all
Attempting git pull for /Users/gpugh/Library/AutoPkg/RecipeRepos/com.github.autopkg.cgerke-recipes...
Already up to date.

Attempting git pull for /Users/gpugh/Library/AutoPkg/RecipeRepos/com.github.autopkg.jss-recipes...
Updating 608ea78..d79e5b5
Fast-forward
 iTerm2/iTerm2.jss.recipe                             |  4 +
 1 file changed, 4 insertions(+)

Attempting git pull for /Users/gpugh/Library/AutoPkg/RecipeRepos/com.github.autopkg.andrewvalentine-
recipes...
Updating b6d4d9e..a025f7d
Fast-forward
 .../EndNoteX9-self-install.pkg.recipe              | 55 ++++++++++++++++++++++
 EndNote-self-install/scripts/postinstall           |  5 ++
 2 files changed, 60 insertions(+)
 create mode 100644 EndNote-self-install/EndNoteX9-self-install.pkg.recipe
 create mode 100755 EndNote-self-install/scripts/postinstall

Attempting git pull for /Users/gpugh/Library/AutoPkg/RecipeRepos/com.github.autopkg.dataJAR-recipes...
Updating 2028422..34ab82f
Fast-forward
 Apache NetBeans 11/Apache NetBeans 11.munki.recipe |   4 -
```

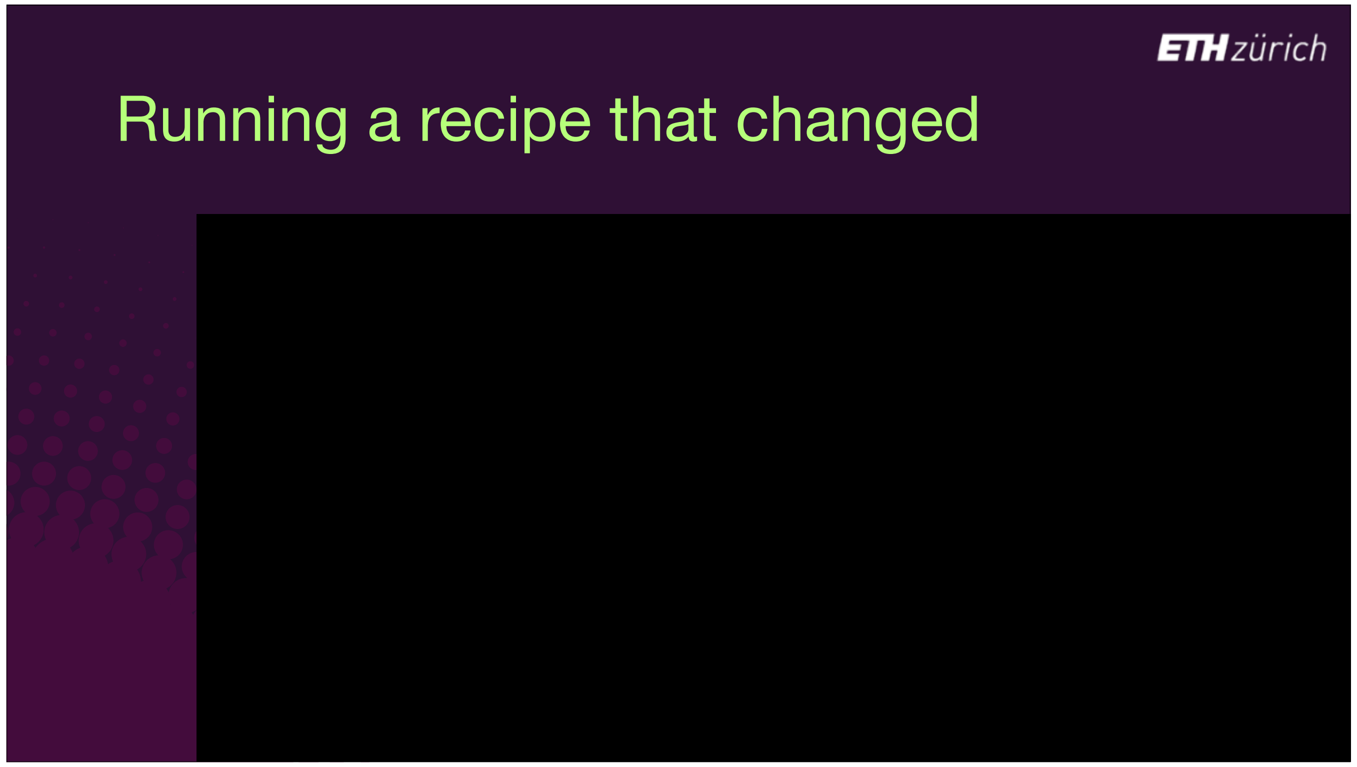


Running a recipe that changed

- Let's run that recipe again.
- It's going to fail, because the contents differ from expected - the hash of the file in the repo does not now match the hash in your override.
- At this point you want to go look at the recipe again and see what changed.

Running a recipe that changed

```
$ autopkg run iTerm2.jss

Failed local trust verification.

The following recipes failed:
    iTerm2.jss
        Parent recipe com.github.jss-recipes.jss.iTerm2 contents differ from expected.
            Path: /Users/gpugh/Library/AutoPkg/RecipeRepos/com.github.autopkg.jss-recipes/iTerm2/
                  iTerm2.jss.recipe

Nothing downloaded, packaged or imported.
```



Running a recipe that changed

To help make it easy to see what changed, there is the "verify-trust-info" verb.

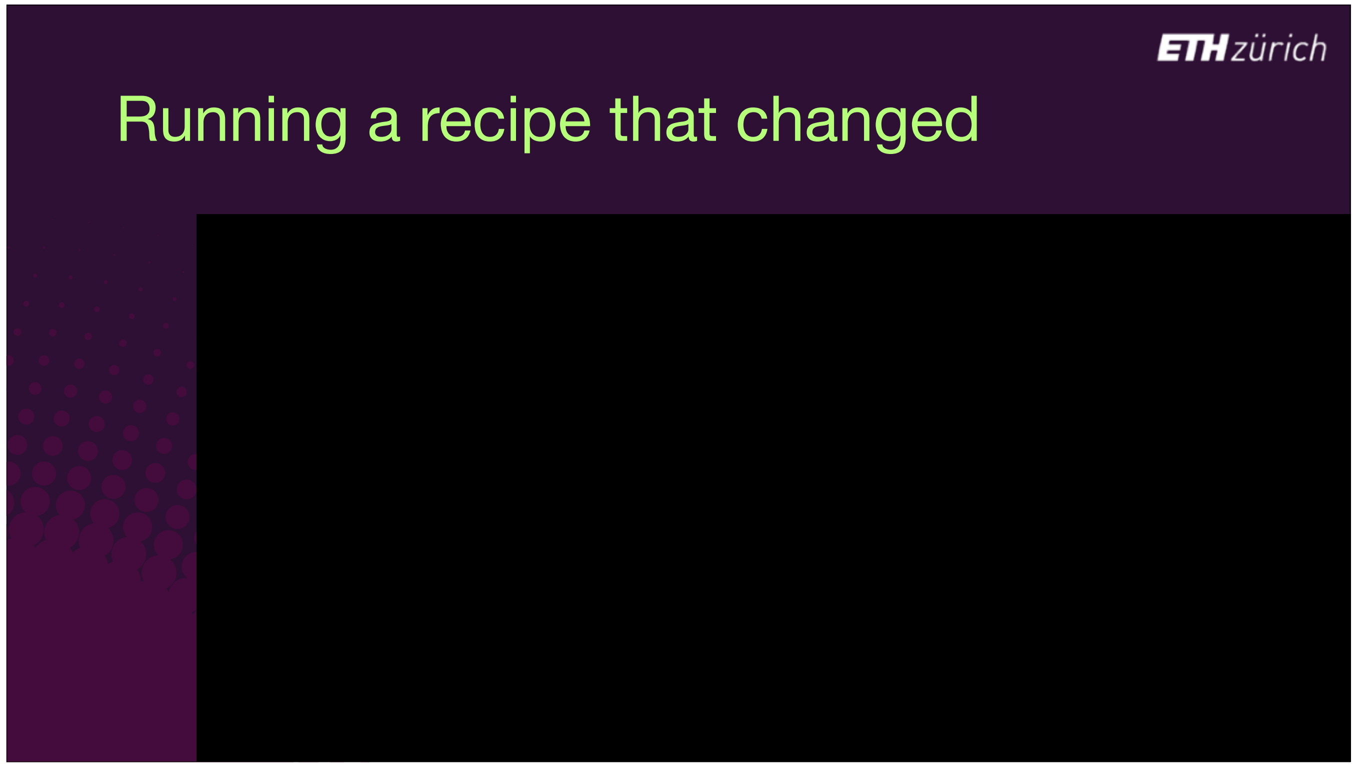
- If you run it with two Vs to get a verbose output,
- [click] it shows you a git diff of the file contents, nicely colour coded.
- This change is innocuous, just a change to the self service description, so I'm happy to let it run.

Running a recipe that changed

```
$ autopkg verify-trust-info -vv iTerm2.jss
```

Running a recipe that changed

```
$ autopkg verify-trust-info -vv iTerm2.jss

iTerm2.jss: FAILED
    Parent recipe com.github.jss-recipes.jss.iTerm2 contents differ from expected.
        Path: /Users/gpugh/Library/AutoPkg/RecipeRepos/com.github.autopkg.jss-recipes/
iTerm2/iTerm2.jss.recipe
    diff --git a/iTerm2/iTerm2.jss.recipe b/iTerm2/iTerm2.jss.recipe
    index 36a1c78..f9c081a 100644
    --- a/iTerm2/iTerm2.jss.recipe
    +++ b/iTerm2/iTerm2.jss.recipe
    @@ -23,7 +23,7 @@
       <key>POLICY_TEMPLATE</key>
       <string>PolicyTemplate.xml</string>
       <key>SELF_SERVICE_DESCRIPTION</key>
    -  <string>iTerm2 is a replacement for Terminal.</string>
    +  <string>iTerm2 is a free alternative to Terminal.</string>
       <key>SELF_SERVICE_ICON</key>
```



Running a recipe that changed

Once you're happy that you trust the changes, enter the "update-trust-info" verb and your override is updated with the new hashes, ready to run the recipe again.

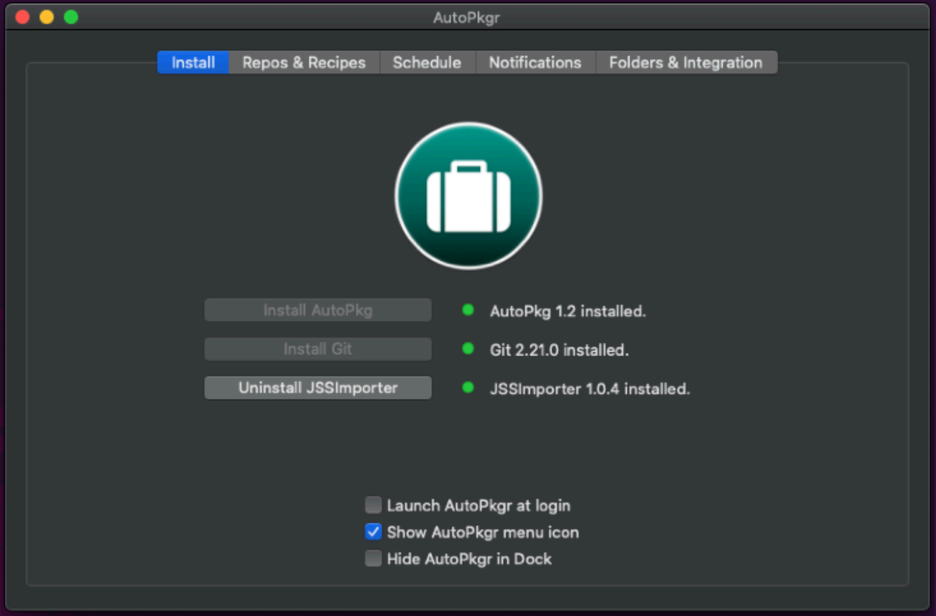
ETH zürich

Running a recipe that changed

```
$ autopkg update-trust-info iTerm2.jss

Wrote updated /Users/gpugh/Library/AutoPkg/RecipeOverrides/iTerm2.jss.recipe
```

- There's also a great Mac application that you can use, that works on top of AutoPkg, called AutoPkgr.
- [click] This can be used to everything from installing and configuring Git, AutoPkg and JSSImporter, performing all the commands I have been describing via the GUI, and sending notifications to slack, email and so on.
- The app was dormant for a while, but Shawn Honsburger at LindeGroup has now updated it for Catalina and it's working with JSSImporter again - A huge thank you to Shawn for bringing AutoPkgr back to life!
- It is important to understand that AutoPkgr is a wrapper on top of AutoPkg - when you do something in the GUI, it's just running the commands.
- So even if you use AutoPkgr, you should know the autopkg commands, and run them in verbose mode for troubleshooting problems.

ETH zürich



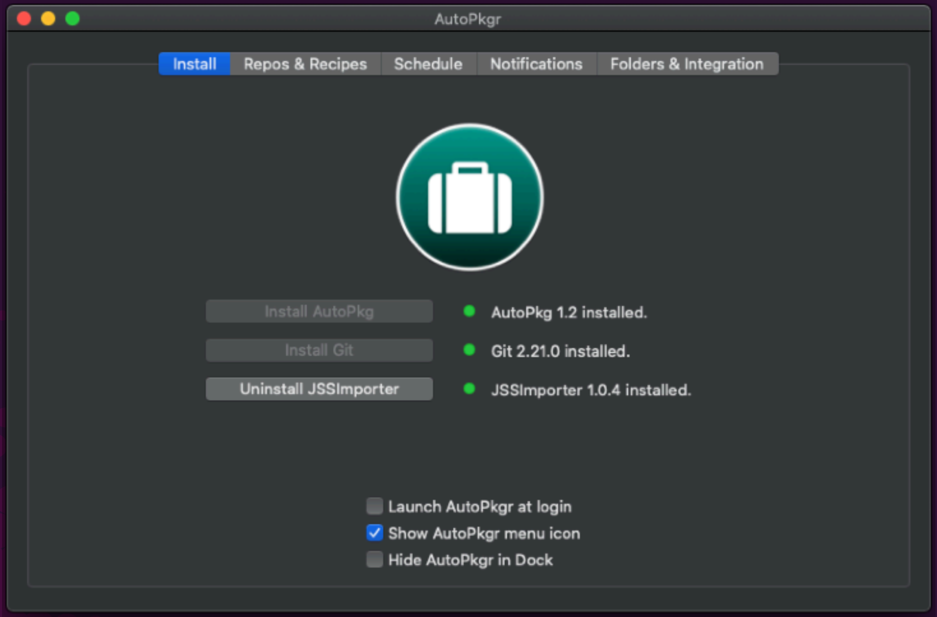
● AutoPkg 1.2 installed.

● Git 2.21.0 installed.

● JSSImporter 1.0.4 installed.

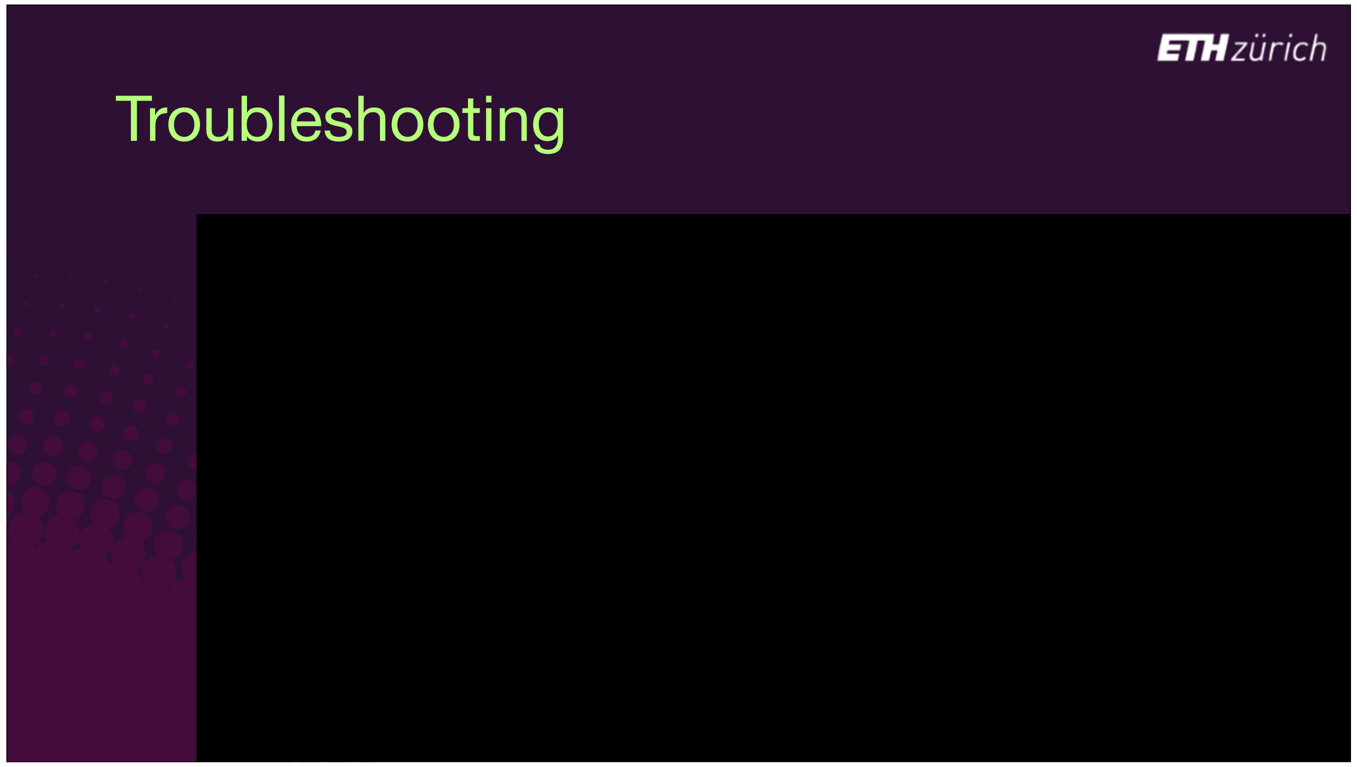
- [ ] Launch AutoPkgr at login
- [x] Show AutoPkgr menu icon
- [ ] Hide AutoPkgr in Dock

ETH zürich



● AutoPkg 1.2 installed.
● Git 2.21.0 installed.
● JSSImporter 1.0.4 installed.

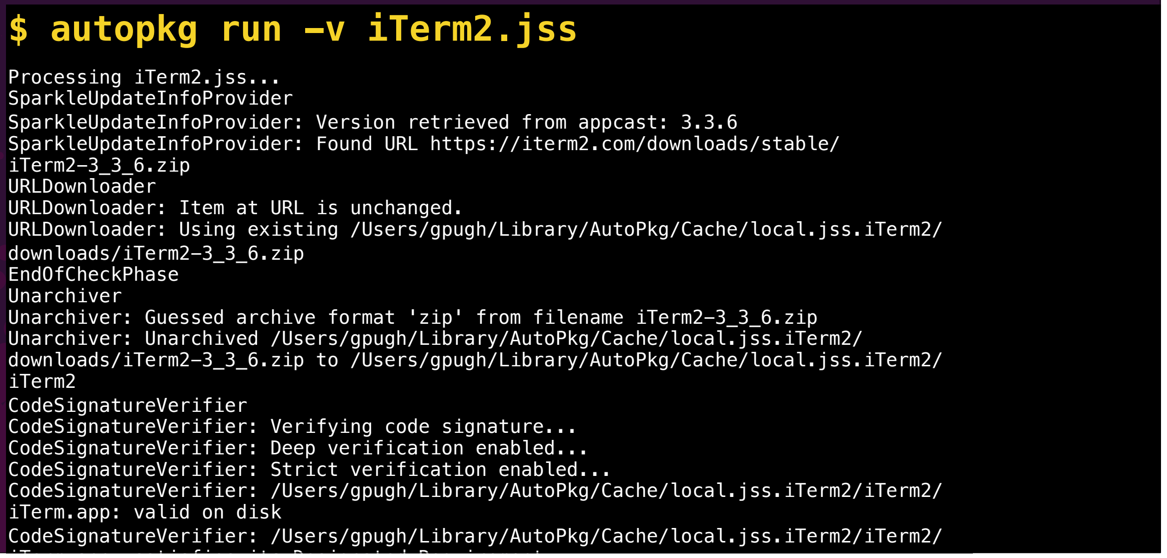
Launch AutoPkgr at login
[✓ Show AutoPkgr menu icon]
Hide AutoPkgr in Dock


Troubleshooting

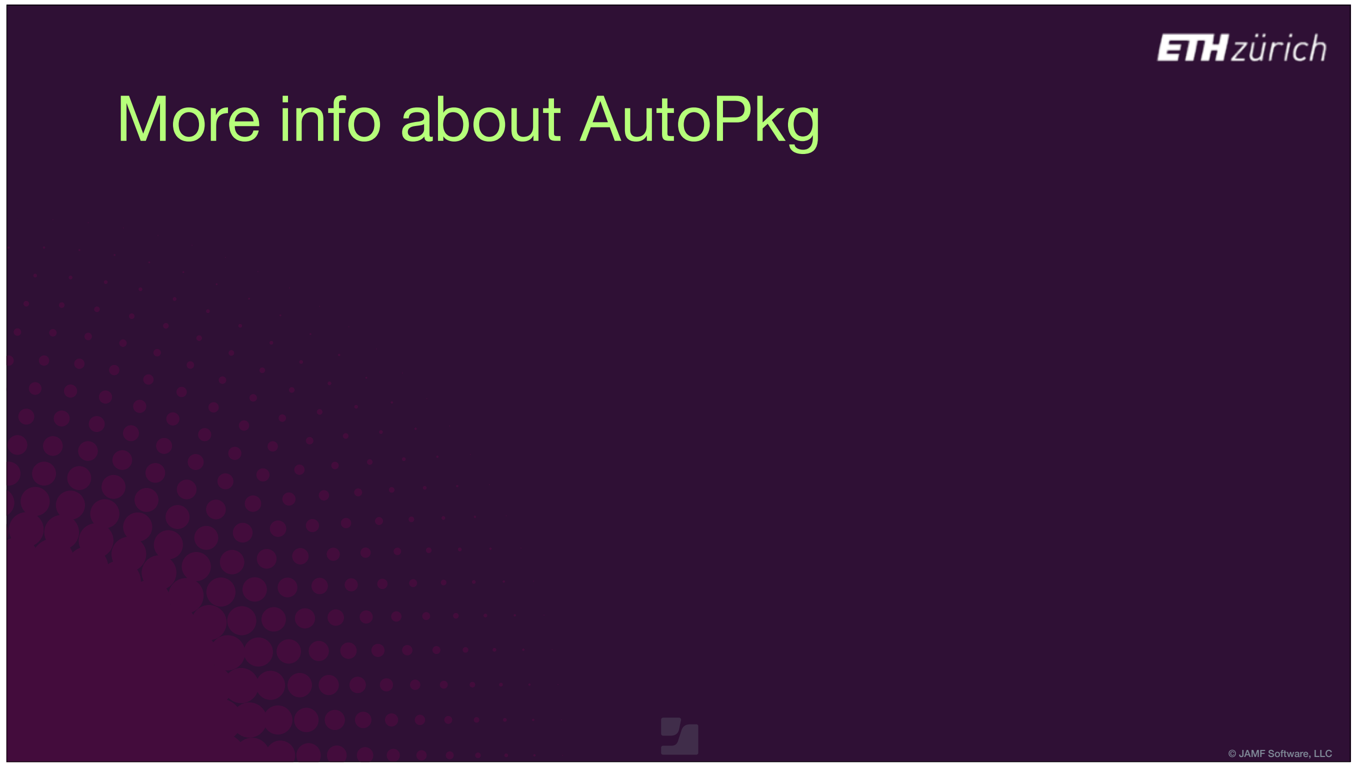
- For example, if we run a recipe with -v, we get much more verbosity about what's going on - you see output of each processor in turn.
- You can add up to 4 Vs to get extremely detailed output.

ETH zürich

Troubleshooting



```
$ autopkg run -v iTerm2.jss
Processing iTerm2.jss...
SparkleUpdateInfoProvider
SparkleUpdateInfoProvider: Version retrieved from appcast: 3.3.6
SparkleUpdateInfoProvider: Found URL https://iterm2.com/downloads/stable/
iTerm2-3_3_6.zip
URLDownloader
URLDownloader: Item at URL is unchanged.
URLDownloader: Using existing /Users/gpugh/Library/AutoPkg/Cache/local.jss.iTerm2/
downloads/iTerm2-3_3_6.zip
EndOfCheckPhase
Unarchiver
Unarchiver: Guessed archive format 'zip' from filename iTerm2-3_3_6.zip
Unarchiver: Unarchived /Users/gpugh/Library/AutoPkg/Cache/local.jss.iTerm2/
downloads/iTerm2-3_3_6.zip to /Users/gpugh/Library/AutoPkg/Cache/local.jss.iTerm2/
iTerm2
CodeSignatureVerifier
CodeSignatureVerifier: Verifying code signature...
CodeSignatureVerifier: Deep verification enabled...
CodeSignatureVerifier: Strict verification enabled...
CodeSignatureVerifier: /Users/gpugh/Library/AutoPkg/Cache/local.jss.iTerm2/iTerm2/
iTerm.app: valid on disk
CodeSignatureVerifier: /Users/gpugh/Library/AutoPkg/Cache/local.jss.iTerm2/iTerm2/
```

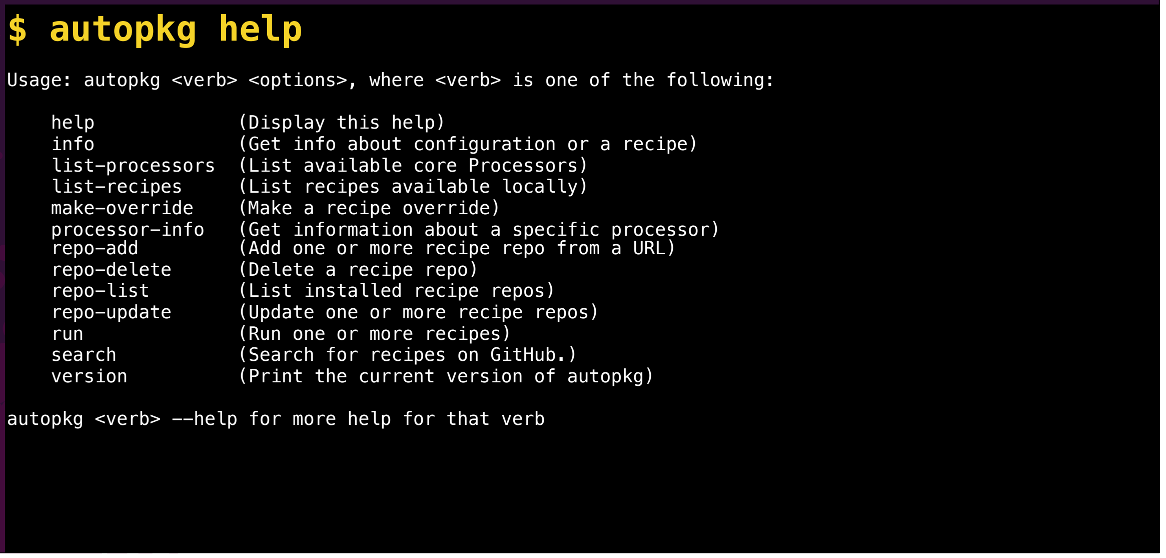


More info about AutoPkg

© JAMF Software, LLC

- To find out more, the autopkg 'help' verb gives you a complete set of options.
- [click] Also, the AutoPkg wiki on GitHub is an essential resource for understanding AutoPkg.

More info about AutoPkg



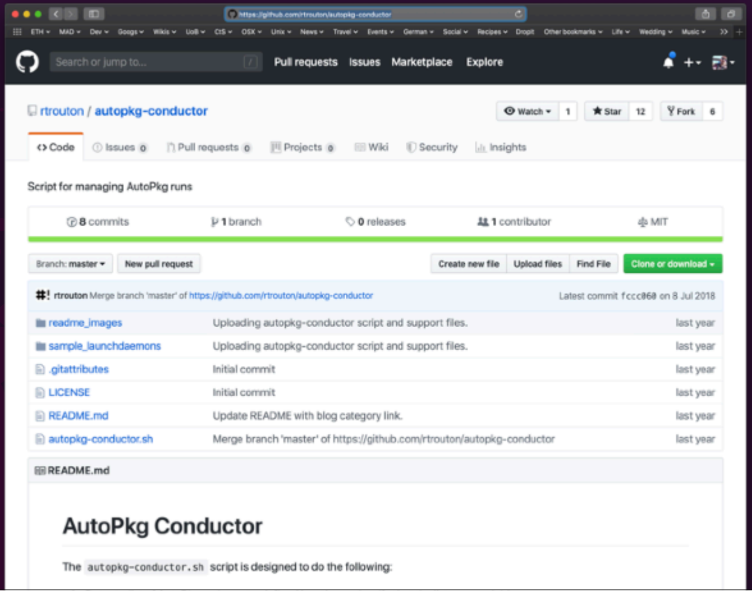
```
$ autopkg help

Usage: autopkg <verb> <options>, where <verb> is one of the following:

    help             (Display this help)
    info             (Get info about configuration or a recipe)
    list-processors  (List available core Processors)
    list-recipes     (List recipes available locally)
    make-override    (Make a recipe override)
    processor-info   (Get information about a specific processor)
    repo-add         (Add one or more recipe repo from a URL)
    repo-delete      (Delete a recipe repo)
    repo-list        (List installed recipe repos)
    repo-update      (Update one or more recipe repos)
    run              (Run one or more recipes)
    search           (Search for recipes on GitHub.)
    version          (Print the current version of autopkg)

autopkg <verb> --help for more help for that verb
```

ETH zürich

More info about AutoPkg

github.com/autopkg/autopkg/wiki

AutoPkg Conductor



rtrouton / autopkg-conductor

Script for managing AutoPkg runs

8 commits | 1 branch | 0 releases | 1 contributor | MIT

rtrouton Merge branch 'master' of https://github.com/rtrouton/autopkg-conductor — Latest commit fccc868 on 8 Jul 2018

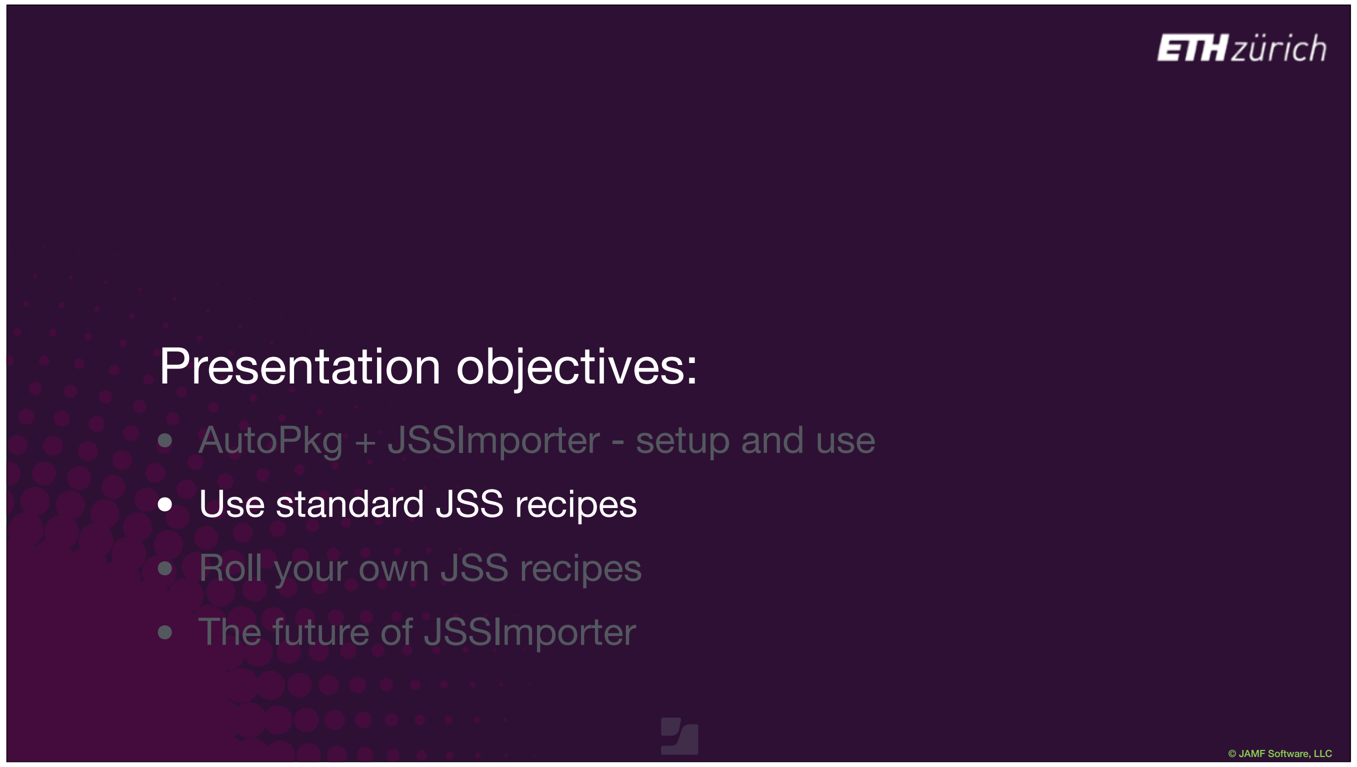
readme_images	Uploading autopkg-conductor script and support files.	last year
sample_launchdaemons	Uploading autopkg-conductor script and support files.	last year
.gitattributes	Initial commit	last year
LICENSE	Initial commit	last year
README.md	Update README with blog category link.	last year
autopkg-conductor.sh	Merge branch 'master' of https://github.com/rtrouton/autopkg-conductor	last year

README.md

AutoPkg Conductor

The `autopkg-conductor.sh` script is designed to do the following:

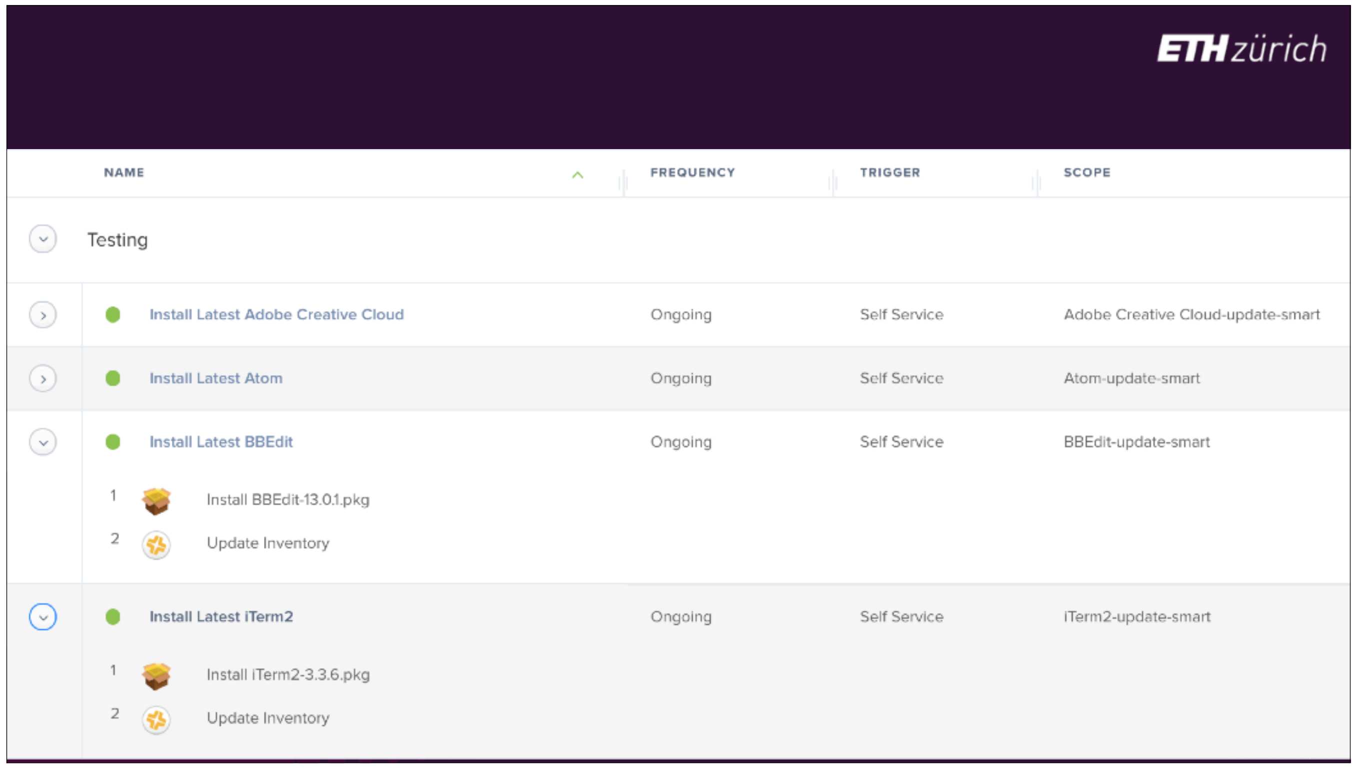
- [slide was not shown in the session]
- Rich Trouton has a script which helps you automate the running of AutoPkg and provide notifications to Slack.



Presentation objectives:

- AutoPkg + JSSImporter - setup and use
- Use standard JSS recipes
- Roll your own JSS recipes
- The future of JSSImporter

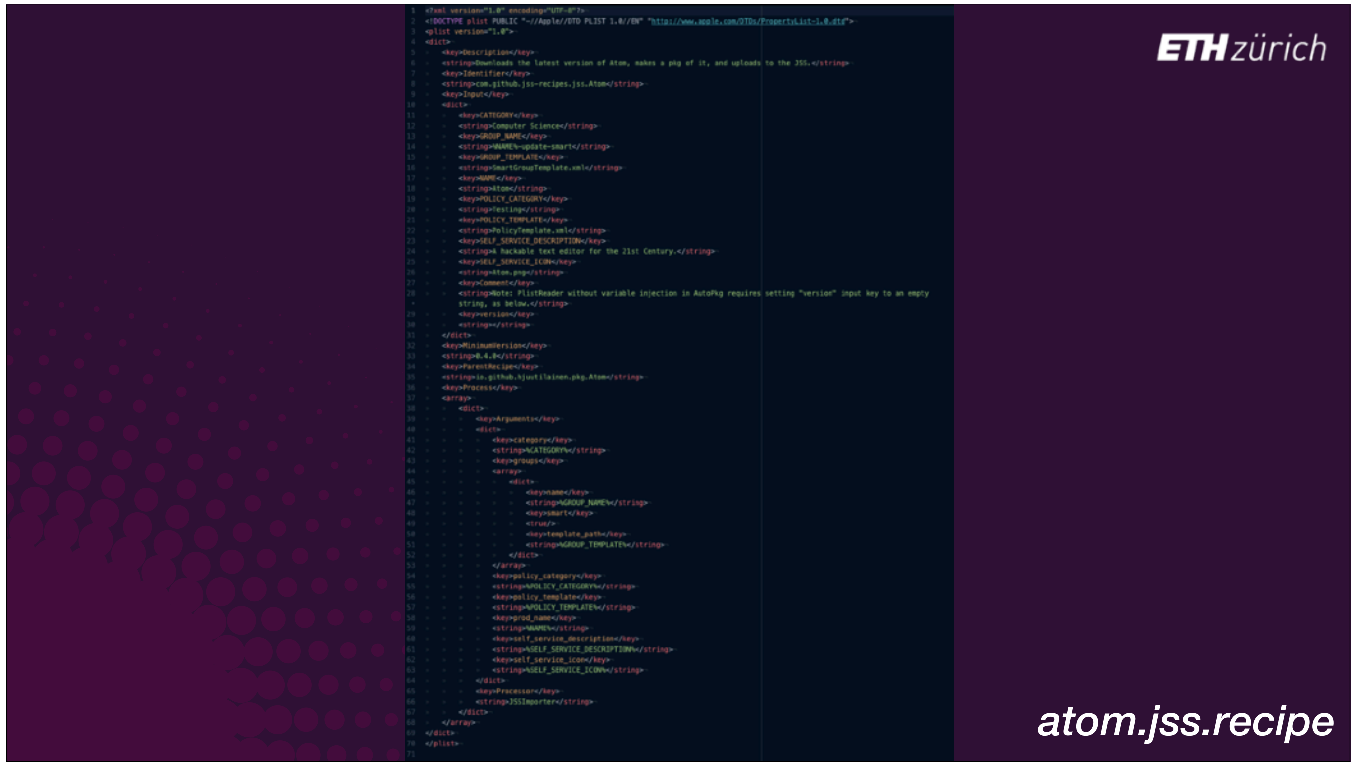
So that was how to use AutoPkg.
Let's look inside the standard JSS recipes, to try an understand what is going on under the hood, and see what we can customise.



	NAME	FREQUENCY	TRIGGER	SCOPE
	Testing			
●	Install Latest Adobe Creative Cloud	Ongoing	Self Service	Adobe Creative Cloud-update-smart
●	Install Latest Atom	Ongoing	Self Service	Atom-update-smart
●	Install Latest BBEdit	Ongoing	Self Service	BBEdit-update-smart
1	Install BBEdit-13.0.1.pkg			
2	Update Inventory			
●	Install Latest iTerm2	Ongoing	Self Service	iTerm2-update-smart
1	Install iTerm2-3.3.6.pkg			
2	Update Inventory			

JSS recipes are designed to import packages and create testing policies.

- If we look at a Jamf Pro server where a few of the standard JSS recipes have run,
- we see that each recipe provides consistent content in your server:
    - A Self Service policy in a Testing category,
    - with ongoing frequency,
    - with the package attached, updating inventory,
    - and a smart group for each policy with a consistent name.



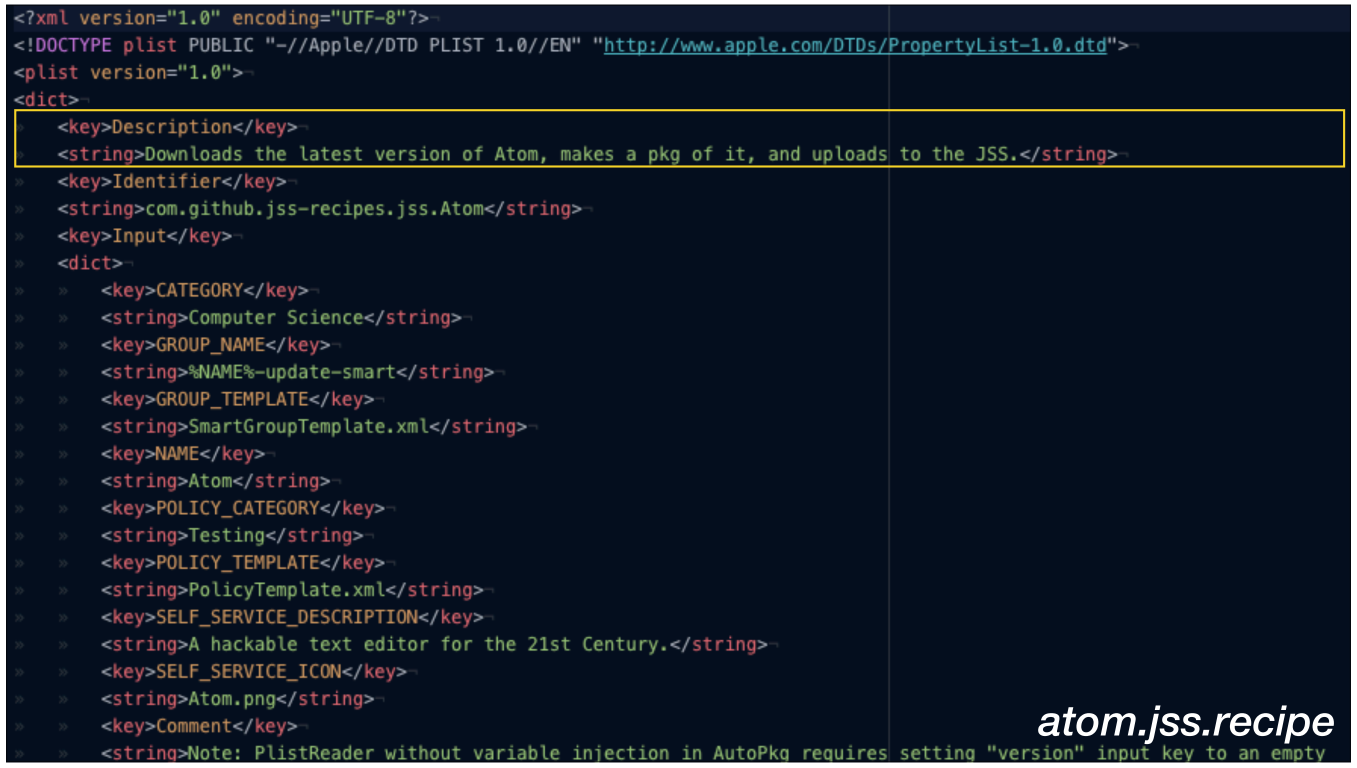
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>Description</key>
    <string>Downloads the latest version of Atom, makes a pkg of it, and uploads to the JSS.</string>
    <key>Identifier</key>
    <string>com.github.jss-recipes.jss.Atom</string>
    <key>Input</key>
    <dict>
        <key>CATEGORY</key>
        <string>Computer Science</string>
        <key>GROUP_NAME</key>
        <string>%NAME%-update-smart</string>
        <key>GROUP_TEMPLATE</key>
        <string>SmartGroupTemplate.xml</string>
        <key>NAME</key>
        <string>Atom</string>
        <key>POLICY_CATEGORY</key>
        <string>Testing</string>
        <key>POLICY_TEMPLATE</key>
        <string>PolicyTemplate.xml</string>
        <key>SELF_SERVICE_DESCRIPTION</key>
        <string>A hackable text editor for the 21st Century.</string>
        <key>SELF_SERVICE_ICON</key>
        <string>Atom.png</string>
        <key>Comment</key>
        <string>Note: PlistReader without variable injection in AutoPkg requires setting "version" input key to an empty
        string, as below.</string>
        <key>version</key>
        <string></string>
    </dict>
    <key>MinimumVersion</key>
    <string>0.4.0</string>
    <key>ParentRecipe</key>
    <string>io.github.hjuutilainen.pkg.Atom</string>
    <key>Process</key>
    <array>
        <dict>
            <key>Arguments</key>
            <dict>
                <key>category</key>
                <string>%CATEGORY%</string>
                <key>groups</key>
                <array>
                    <dict>
                        <key>name</key>
                        <string>%GROUP_NAME%</string>
                        <key>smart</key>
                        <true/>
                        <key>template_path</key>
                        <string>%GROUP_TEMPLATE%</string>
                    </dict>
                </array>
                <key>policy_category</key>
                <string>%POLICY_CATEGORY%</string>
                <key>policy_template</key>
                <string>%POLICY_TEMPLATE%</string>
                <key>prod_name</key>
                <string>%NAME%</string>
                <key>self_service_description</key>
                <string>%SELF_SERVICE_DESCRIPTION%</string>
                <key>self_service_icon</key>
                <string>%SELF_SERVICE_ICON%</string>
            </dict>
            <key>Processor</key>
            <string>JSSImporter</string>
        </dict>
    </array>
</dict>
</plist>
```

ETH zürich

atom.jss.recipe

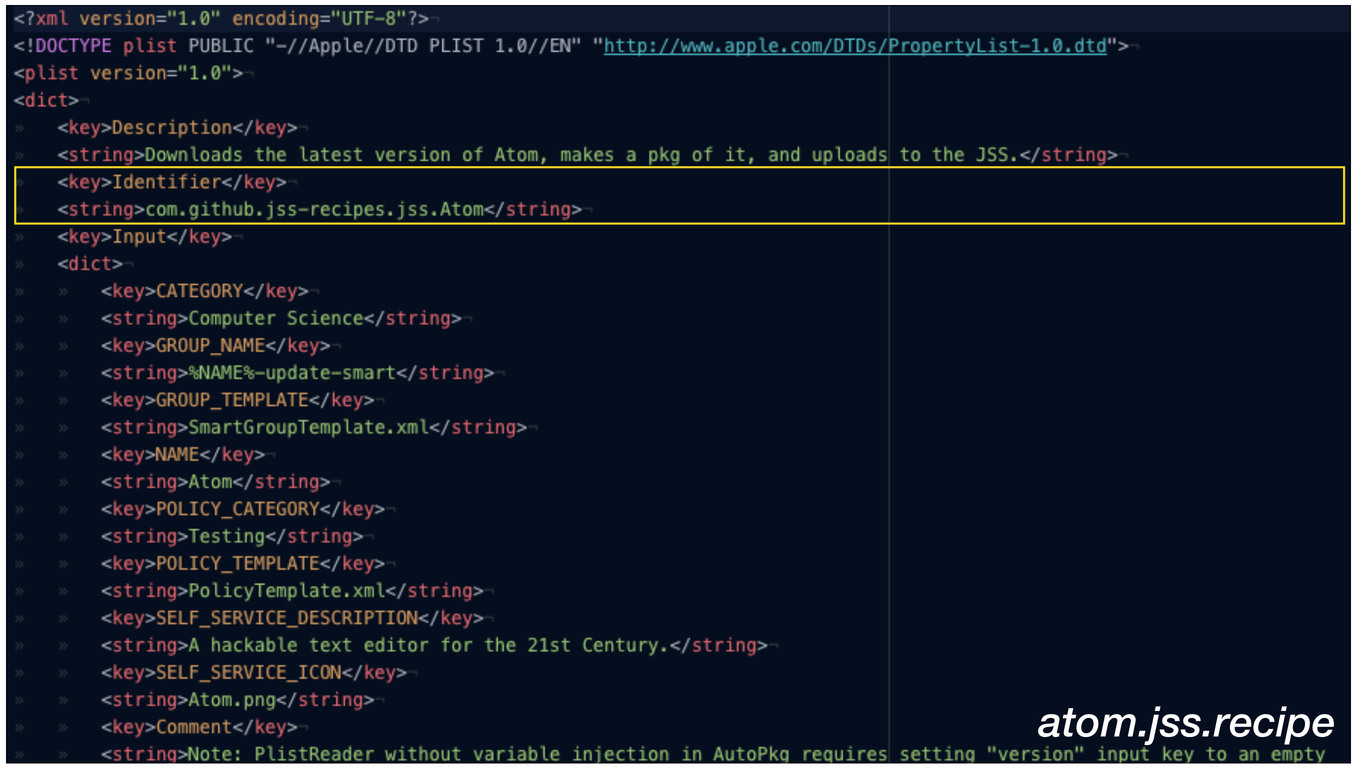
Let's look at a standard JSS Recipe.

- Here's the Atom.jss recipe. It's a PLIST file, which aren't the nicest things to look at – but let's zoom in and break it down.
- The description, and [click] recipe identifier are at the top.
- [click] And further down, the reference to the Parent PKG Recipe.
- [click] Underneath that we have the information about the JSSImporter process itself.
    - There are keys for the package and policy categories, Smart Group and policy templates, policy name, self service description and icon.
- Each key has a value inside percent signs. These are overridable variables, and you'll see them in all AutoPkg recipes.
- [click] At the top, the Input section is where the default values for those overridable variables are set.
- If you want to change any of these overridable values, you don't edit the JSS recipe itself. You instead go to the recipe override file.



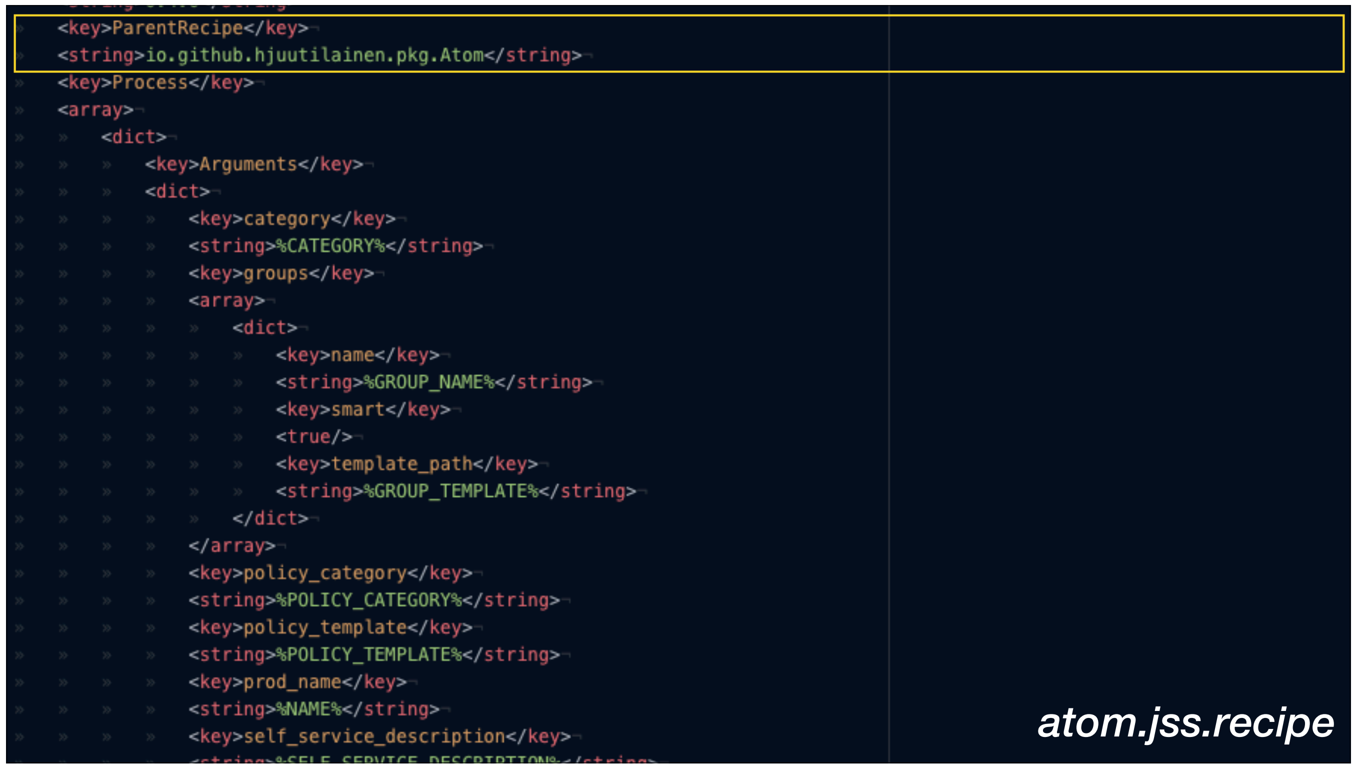
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>Description</key>
    <string>Downloads the latest version of Atom, makes a pkg of it, and uploads to the JSS.</string>
    <key>Identifier</key>
    <string>com.github.jss-recipes.jss.Atom</string>
    <key>Input</key>
    <dict>
        <key>CATEGORY</key>
        <string>Computer Science</string>
        <key>GROUP_NAME</key>
        <string>%NAME%-update-smart</string>
        <key>GROUP_TEMPLATE</key>
        <string>SmartGroupTemplate.xml</string>
        <key>NAME</key>
        <string>Atom</string>
        <key>POLICY_CATEGORY</key>
        <string>Testing</string>
        <key>POLICY_TEMPLATE</key>
        <string>PolicyTemplate.xml</string>
        <key>SELF_SERVICE_DESCRIPTION</key>
        <string>A hackable text editor for the 21st Century.</string>
        <key>SELF_SERVICE_ICON</key>
        <string>Atom.png</string>
        <key>Comment</key>
        <string>Note: PlistReader without variable injection in AutoPkg requires setting "version" input key to an empty
```

atom.jss.recipe



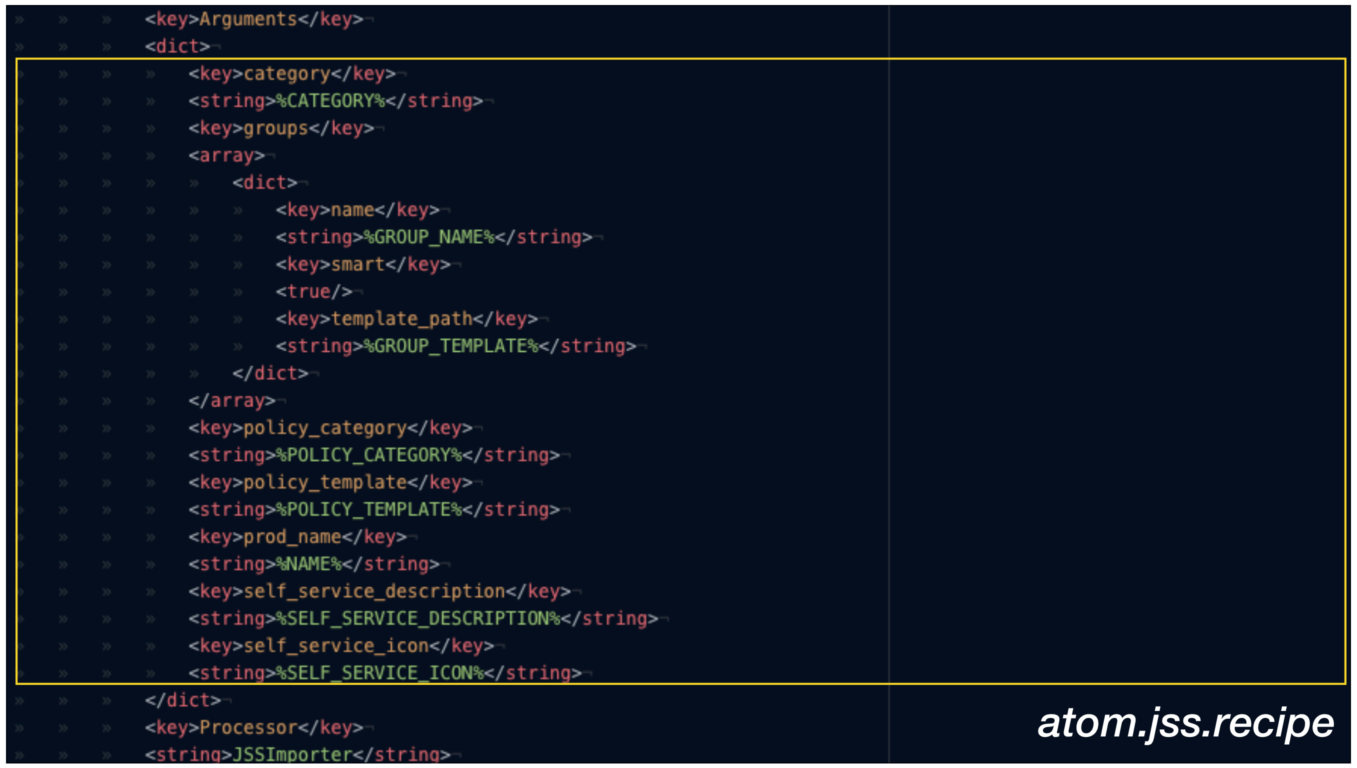
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>Description</key>
    <string>Downloads the latest version of Atom, makes a pkg of it, and uploads to the JSS.</string>
    <key>Identifier</key>
    <string>com.github.jss-recipes.jss.Atom</string>
    <key>Input</key>
    <dict>
        <key>CATEGORY</key>
        <string>Computer Science</string>
        <key>GROUP_NAME</key>
        <string>%NAME%-update-smart</string>
        <key>GROUP_TEMPLATE</key>
        <string>SmartGroupTemplate.xml</string>
        <key>NAME</key>
        <string>Atom</string>
        <key>POLICY_CATEGORY</key>
        <string>Testing</string>
        <key>POLICY_TEMPLATE</key>
        <string>PolicyTemplate.xml</string>
        <key>SELF_SERVICE_DESCRIPTION</key>
        <string>A hackable text editor for the 21st Century.</string>
        <key>SELF_SERVICE_ICON</key>
        <string>Atom.png</string>
        <key>Comment</key>
        <string>Note: PlistReader without variable injection in AutoPkg requires setting "version" input key to an empty
```

atom.jss.recipe



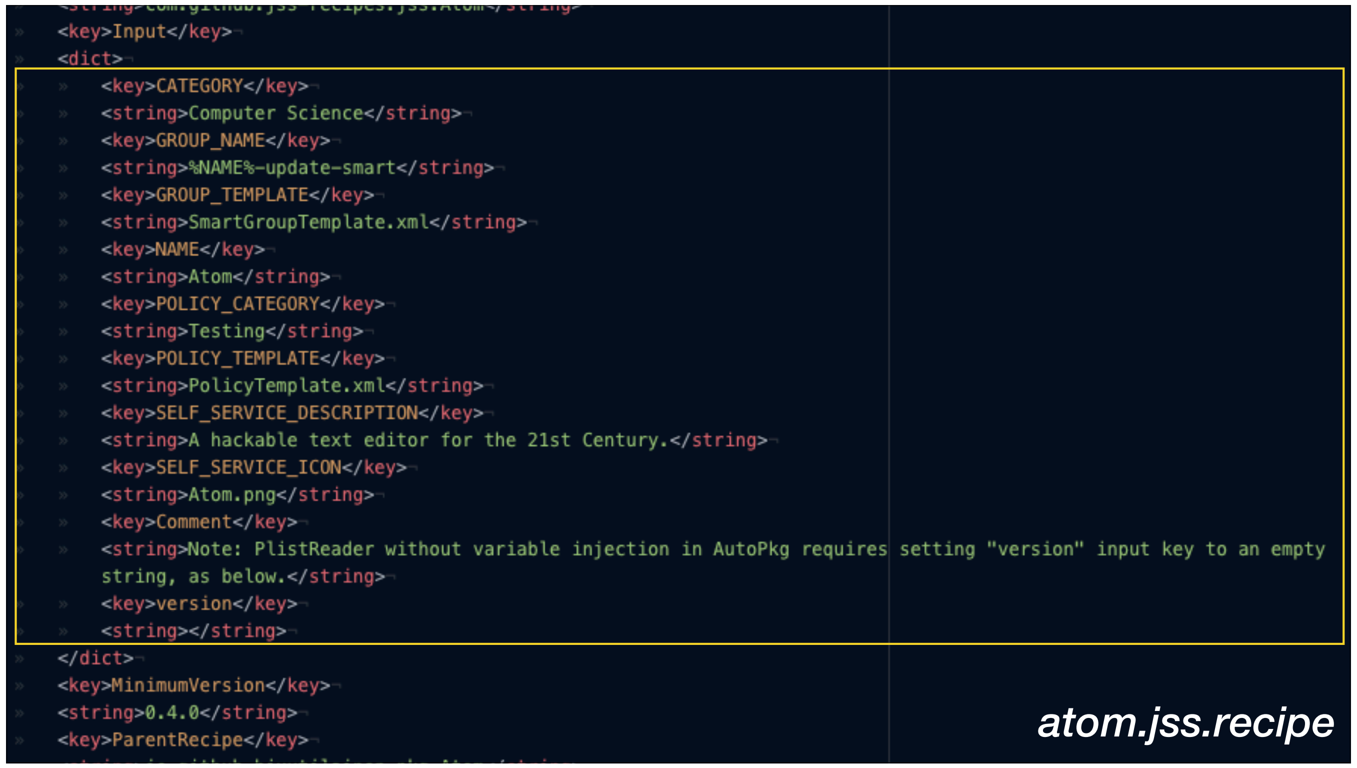
```
    <key>ParentRecipe</key>
    <string>io.github.hjuutilainen.pkg.Atom</string>
    <key>Process</key>
    <array>
        <dict>
            <key>Arguments</key>
            <dict>
                <key>category</key>
                <string>%CATEGORY%</string>
                <key>groups</key>
                <array>
                    <dict>
                        <key>name</key>
                        <string>%GROUP_NAME%</string>
                        <key>smart</key>
                        <true/>
                        <key>template_path</key>
                        <string>%GROUP_TEMPLATE%</string>
                    </dict>
                </array>
                <key>policy_category</key>
                <string>%POLICY_CATEGORY%</string>
                <key>policy_template</key>
                <string>%POLICY_TEMPLATE%</string>
                <key>prod_name</key>
                <string>%NAME%</string>
                <key>self_service_description</key>
                <string>%SELF_SERVICE_DESCRIPTION%</string>
```

atom.jss.recipe



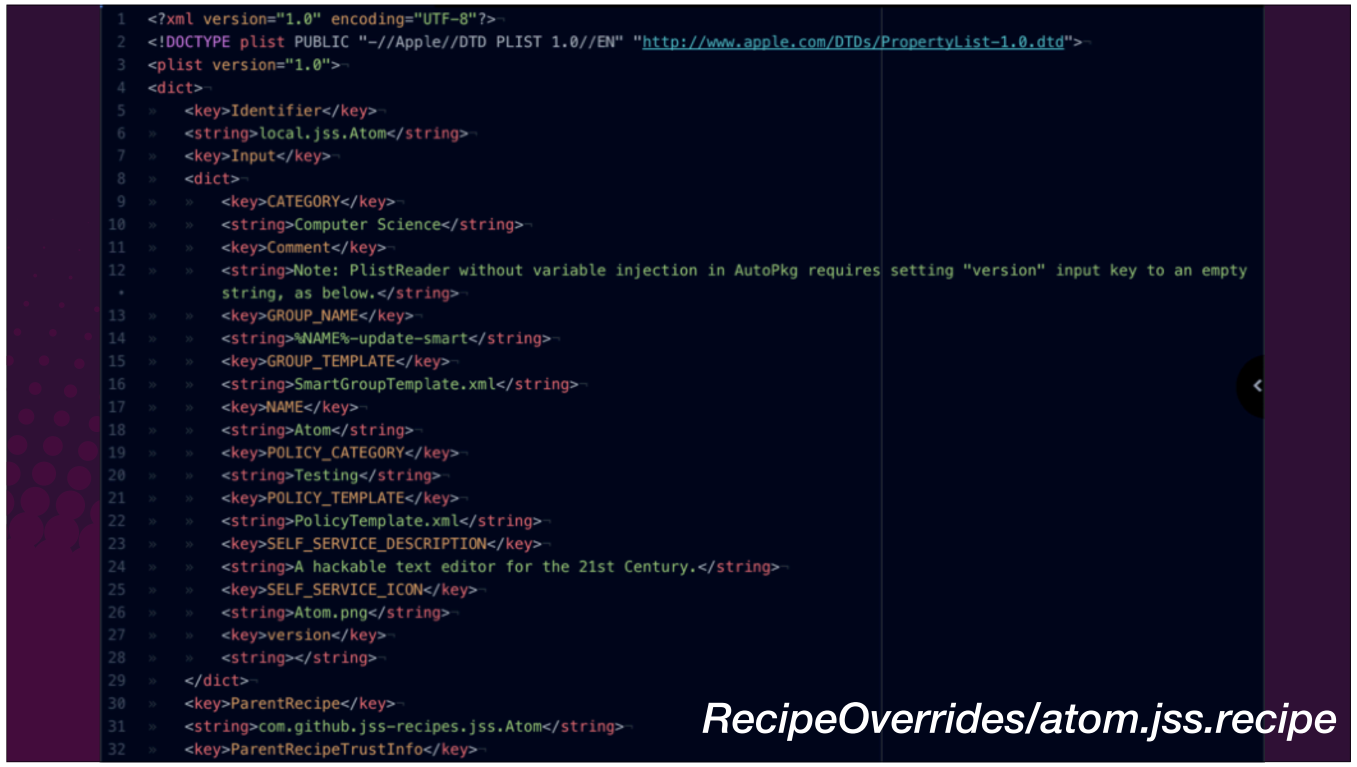
```
			<key>Arguments</key>
			<dict>
				<key>category</key>
				<string>%CATEGORY%</string>
				<key>groups</key>
				<array>
					<dict>
						<key>name</key>
						<string>%GROUP_NAME%</string>
						<key>smart</key>
						<true/>
						<key>template_path</key>
						<string>%GROUP_TEMPLATE%</string>
					</dict>
				</array>
				<key>policy_category</key>
				<string>%POLICY_CATEGORY%</string>
				<key>policy_template</key>
				<string>%POLICY_TEMPLATE%</string>
				<key>prod_name</key>
				<string>%NAME%</string>
				<key>self_service_description</key>
				<string>%SELF_SERVICE_DESCRIPTION%</string>
				<key>self_service_icon</key>
				<string>%SELF_SERVICE_ICON%</string>
			</dict>
			<key>Processor</key>
			<string>JSSImporter</string>
```

atom.jss.recipe



```
    <string>com.github.jss-recipes.jss.Atom</string>
    <key>Input</key>
    <dict>
        <key>CATEGORY</key>
        <string>Computer Science</string>
        <key>GROUP_NAME</key>
        <string>%NAME%-update-smart</string>
        <key>GROUP_TEMPLATE</key>
        <string>SmartGroupTemplate.xml</string>
        <key>NAME</key>
        <string>Atom</string>
        <key>POLICY_CATEGORY</key>
        <string>Testing</string>
        <key>POLICY_TEMPLATE</key>
        <string>PolicyTemplate.xml</string>
        <key>SELF_SERVICE_DESCRIPTION</key>
        <string>A hackable text editor for the 21st Century.</string>
        <key>SELF_SERVICE_ICON</key>
        <string>Atom.png</string>
        <key>Comment</key>
        <string>Note: PlistReader without variable injection in AutoPkg requires setting "version" input key to an empty
        string, as below.</string>
        <key>version</key>
        <string></string>
    </dict>
    <key>MinimumVersion</key>
    <string>0.4.0</string>
    <key>ParentRecipe</key>
```

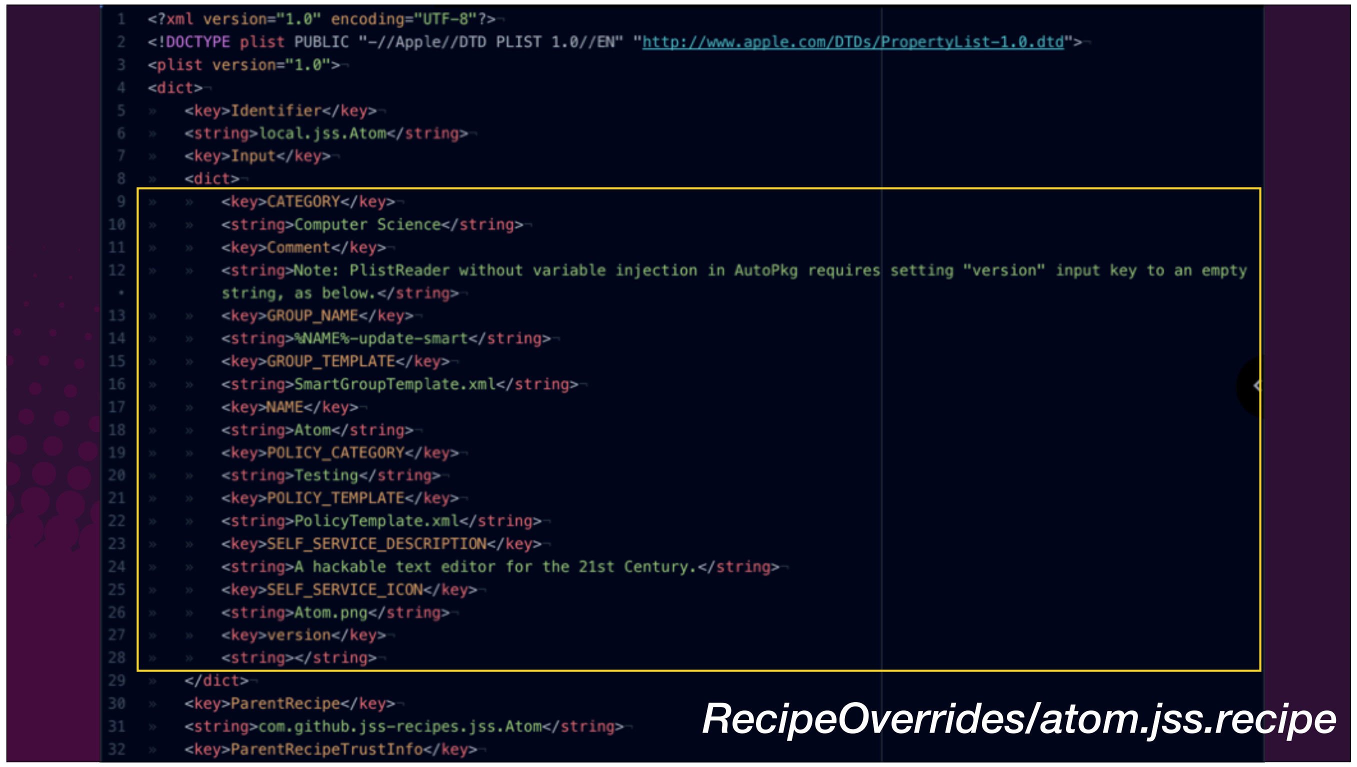
atom.jss.recipe



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>Identifier</key>
    <string>local.jss.Atom</string>
    <key>Input</key>
    <dict>
        <key>CATEGORY</key>
        <string>Computer Science</string>
        <key>Comment</key>
        <string>Note: PlistReader without variable injection in AutoPkg requires setting "version" input key to an empty
        string, as below.</string>
        <key>GROUP_NAME</key>
        <string>%NAME%-update-smart</string>
        <key>GROUP_TEMPLATE</key>
        <string>SmartGroupTemplate.xml</string>
        <key>NAME</key>
        <string>Atom</string>
        <key>POLICY_CATEGORY</key>
        <string>Testing</string>
        <key>POLICY_TEMPLATE</key>
        <string>PolicyTemplate.xml</string>
        <key>SELF_SERVICE_DESCRIPTION</key>
        <string>A hackable text editor for the 21st Century.</string>
        <key>SELF_SERVICE_ICON</key>
        <string>Atom.png</string>
        <key>version</key>
        <string></string>
    </dict>
    <key>ParentRecipe</key>
    <string>com.github.jss-recipes.jss.Atom</string>
    <key>ParentRecipeTrustInfo</key>
```

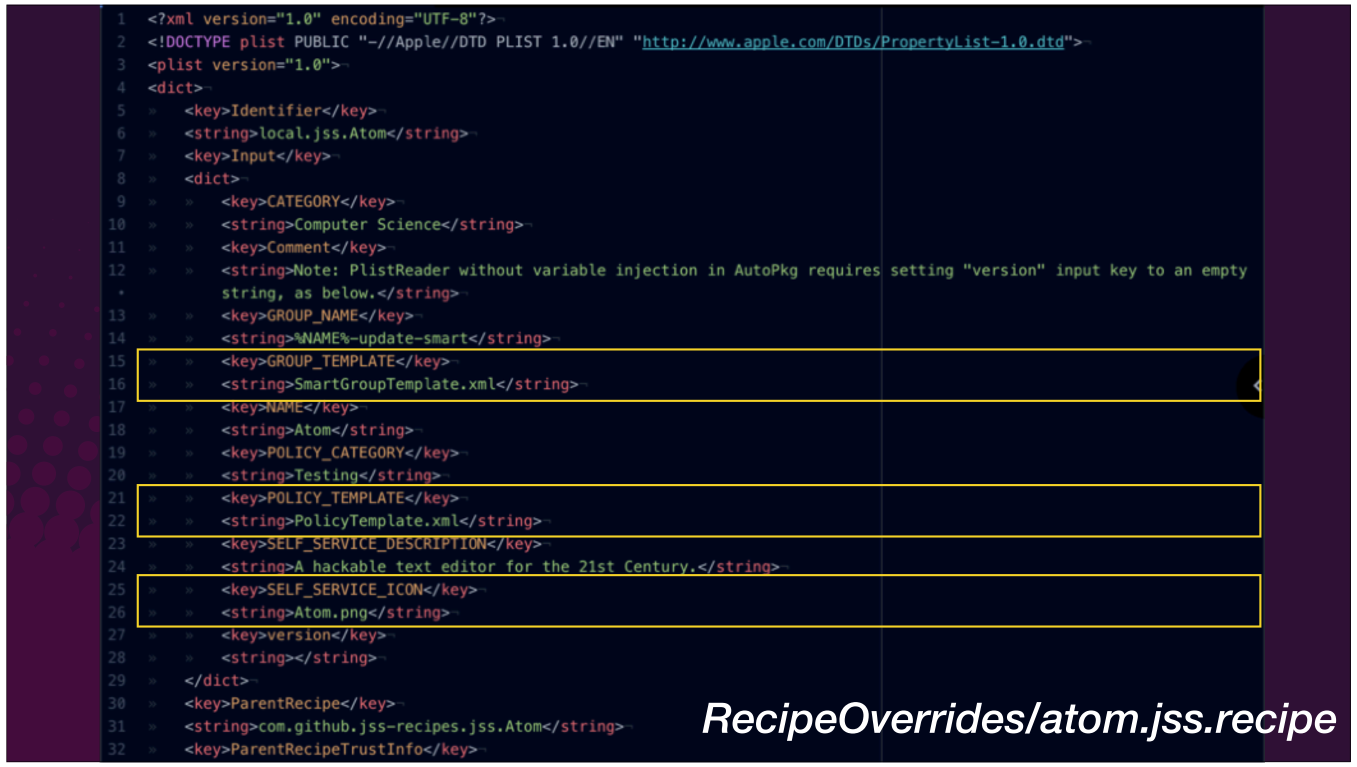
RecipeOverrides/atom.jss.recipe

- Here's the override file for Atom's jss recipe.
- [click] Here are the same Input keys as in the recipe. They are copied here when the override file is made, and if you want to change any value, these values take precedence over the ones in the JSS recipe itself.
- When you update trust info, your changes to the Input keys are not overwritten.
- Notice these three keys for Group Template, Policy Template and Self Service icon. These are referring to separate files.



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
	<key>Identifier</key>
	<string>local.jss.Atom</string>
	<key>Input</key>
	<dict>
		<key>CATEGORY</key>
		<string>Computer Science</string>
		<key>Comment</key>
		<string>Note: PlistReader without variable injection in AutoPkg requires setting "version" input key to an empty
		string, as below.</string>
		<key>GROUP_NAME</key>
		<string>%NAME%-update-smart</string>
		<key>GROUP_TEMPLATE</key>
		<string>SmartGroupTemplate.xml</string>
		<key>NAME</key>
		<string>Atom</string>
		<key>POLICY_CATEGORY</key>
		<string>Testing</string>
		<key>POLICY_TEMPLATE</key>
		<string>PolicyTemplate.xml</string>
		<key>SELF_SERVICE_DESCRIPTION</key>
		<string>A hackable text editor for the 21st Century.</string>
		<key>SELF_SERVICE_ICON</key>
		<string>Atom.png</string>
		<key>version</key>
		<string></string>
	</dict>
	<key>ParentRecipe</key>
	<string>com.github.jss-recipes.jss.Atom</string>
	<key>ParentRecipeTrustInfo</key>
```

RecipeOverrides/atom.jss.recipe



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>Identifier</key>
    <string>local.jss.Atom</string>
    <key>Input</key>
    <dict>
        <key>CATEGORY</key>
        <string>Computer Science</string>
        <key>Comment</key>
        <string>Note: PlistReader without variable injection in AutoPkg requires setting "version" input key to an empty
        string, as below.</string>
        <key>GROUP_NAME</key>
        <string>%NAME%-update-smart</string>
        <key>GROUP_TEMPLATE</key>
        <string>SmartGroupTemplate.xml</string>
        <key>NAME</key>
        <string>Atom</string>
        <key>POLICY_CATEGORY</key>
        <string>Testing</string>
        <key>POLICY_TEMPLATE</key>
        <string>PolicyTemplate.xml</string>
        <key>SELF_SERVICE_DESCRIPTION</key>
        <string>A hackable text editor for the 21st Century.</string>
        <key>SELF_SERVICE_ICON</key>
        <string>Atom.png</string>
        <key>version</key>
        <string></string>
    </dict>
    <key>ParentRecipe</key>
    <string>com.github.jss-recipes.jss.Atom</string>
    <key>ParentRecipeTrustInfo</key>
```

RecipeOverrides/atom.jss.recipe

Standard JSS recipes

- [click] JSS recipes use XML template files to create policy objects and smart groups.
- Like recipes, these files contain variables that can be overridden in the recipe overrides. This means that they can be re-used for multiple recipes – in fact you can use the same ones for most recipes.
- [click] Sometimes recipes also need Extension Attributes or scripts, for scoping, or for post-installation tasks. These scripts and EAs need their own template files.
- These should be provided in the repo of the JSS recipe when required.
- [click] An icon file is also needed, so that the Self Service policy is easily identified. These are also provided in the repository along with the JSS recipe.

Standard JSS recipes

Policy & Group Templates

ETH zürich

Standard JSS recipes

XML

Policy & Group Templates

SH

Extension Attribute & Script Templates

ETH zürich

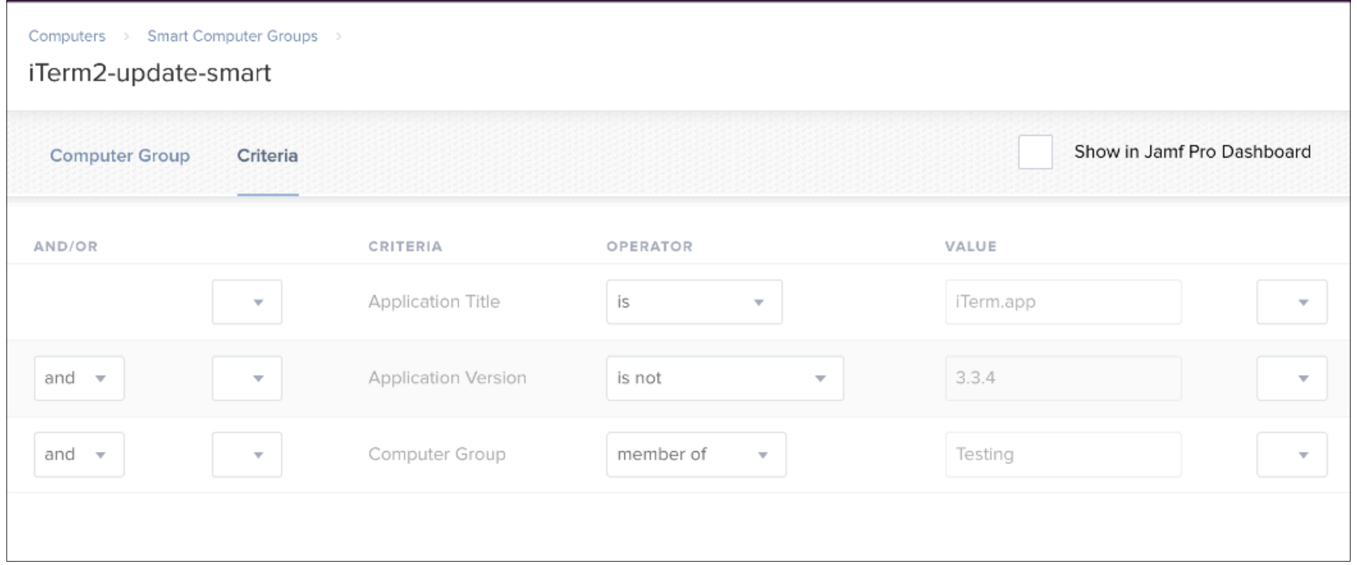
Standard JSS recipes

Policy & Group Templates

Extension Attribute & Script Templates

Self Service icon

Smart Group Templates

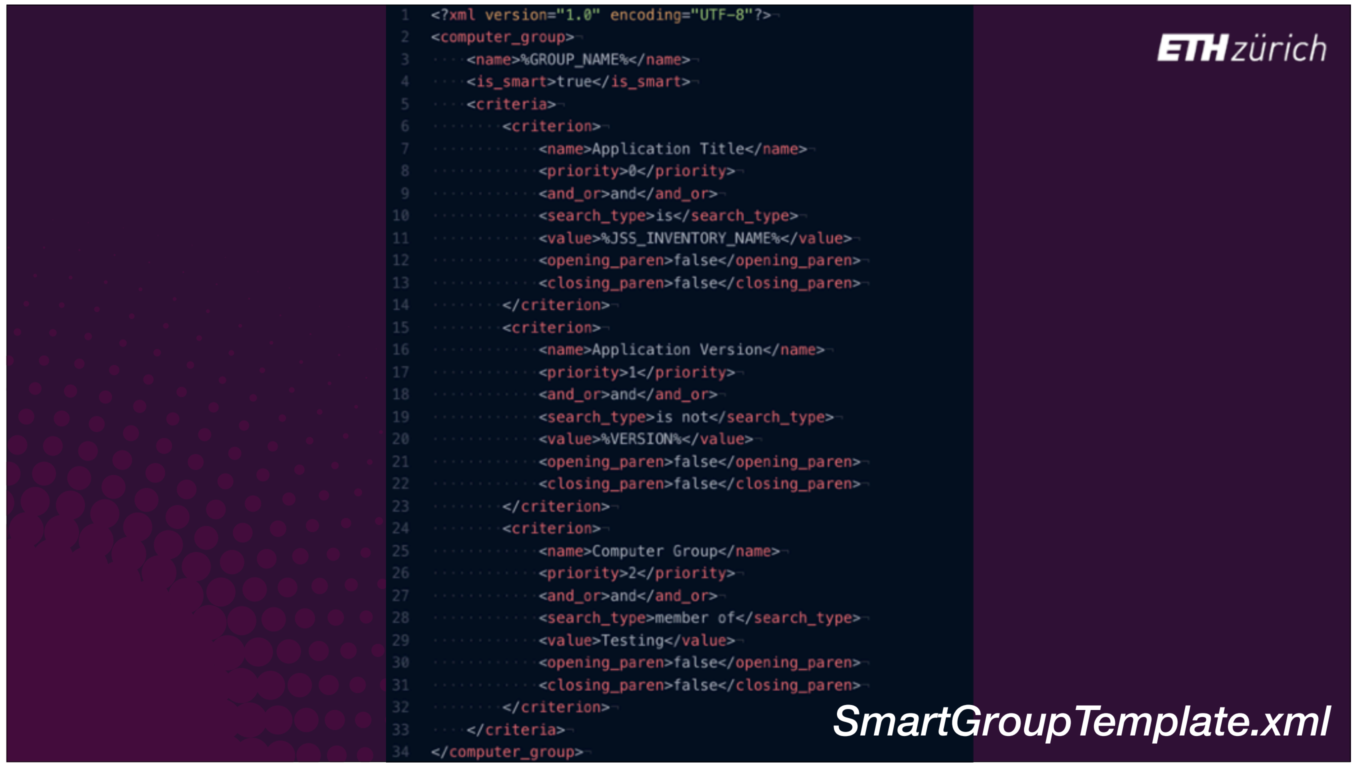

Computers > Smart Computer Groups >

iTerm2-update-smart

AND/OR		CRITERIA	OPERATOR	VALUE	
		Application Title	is	iTerm.app	
and		Application Version	is not	3.3.4	
and		Computer Group	member of	Testing	

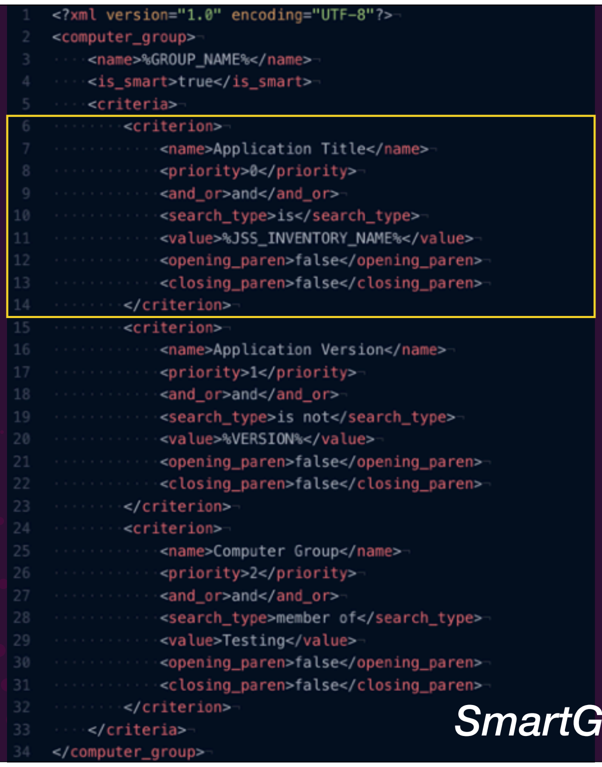
Let's look at the Standard Smart Group created by a JSS recipe.

- The smart group that is created by JSSImporter has a consistent set of criteria.
- The computer has an application of a particular name.
- The application version on the computer does not match the value provided.
- And, the computer is in a group named 'Testing', which you have to make. This can be smart or static.



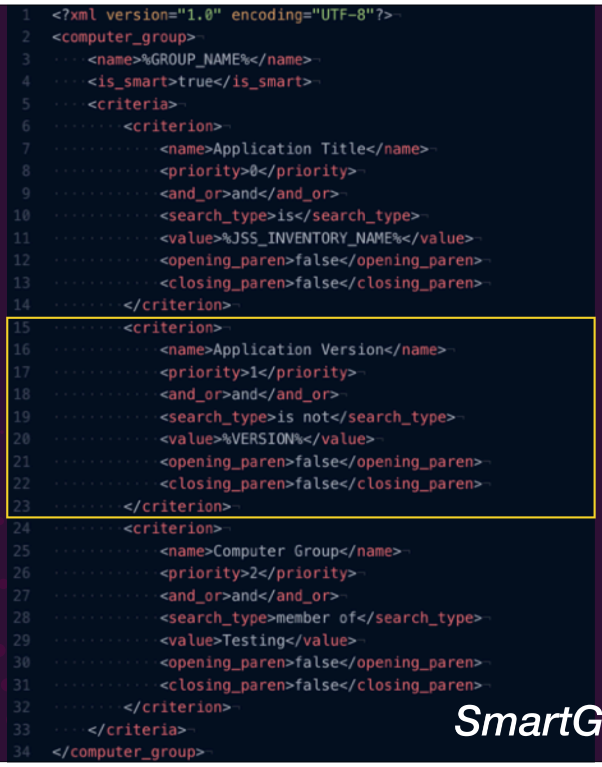
```xml
<?xml version="1.0" encoding="UTF-8"?>
<computer_group>
    <name>%GROUP_NAME%</name>
    <is_smart>true</is_smart>
    <criteria>
        <criterion>
            <name>Application Title</name>
            <priority>0</priority>
            <and_or>and</and_or>
            <search_type>is</search_type>
            <value>%JSS_INVENTORY_NAME%</value>
            <opening_paren>false</opening_paren>
            <closing_paren>false</closing_paren>
        </criterion>
        <criterion>
            <name>Application Version</name>
            <priority>1</priority>
            <and_or>and</and_or>
            <search_type>is not</search_type>
            <value>%VERSION%</value>
            <opening_paren>false</opening_paren>
            <closing_paren>false</closing_paren>
        </criterion>
        <criterion>
            <name>Computer Group</name>
            <priority>2</priority>
            <and_or>and</and_or>
            <search_type>member of</search_type>
            <value>Testing</value>
            <opening_paren>false</opening_paren>
            <closing_paren>false</closing_paren>
        </criterion>
    </criteria>
</computer_group>
```

- If we look in the standard Smart Group Template, you can see how this is generated.
    - [click] The criteria are that the computer has an Application Title matching a title provided in the JSS recipe,
    - [click] the computer has a different version to the one provided in the JSS recipe,
    - [click] And the computer is in a Computer group Testing.
- [click to clear] There's not much you can override in here, as the app name and version are automatically generated and the Testing group is hard coded.
- If you want to add additional criteria, or change to different criteria, you need to change the template directly, so you have to save a copy in your Recipe Overrides folder and then make your edits.



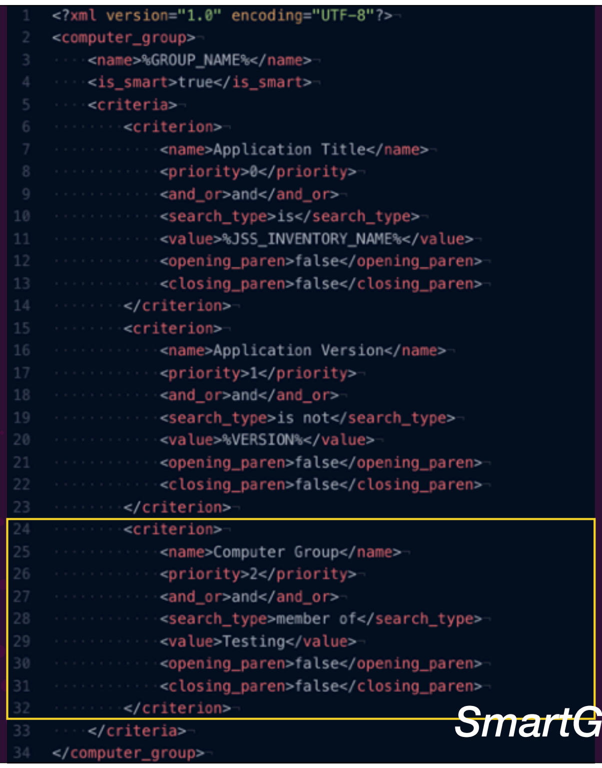
```xml
<?xml version="1.0" encoding="UTF-8"?>
<computer_group>
    <name>%GROUP_NAME%</name>
    <is_smart>true</is_smart>
    <criteria>
        <criterion>
            <name>Application Title</name>
            <priority>0</priority>
            <and_or>and</and_or>
            <search_type>is</search_type>
            <value>%JSS_INVENTORY_NAME%</value>
            <opening_paren>false</opening_paren>
            <closing_paren>false</closing_paren>
        </criterion>
        <criterion>
            <name>Application Version</name>
            <priority>1</priority>
            <and_or>and</and_or>
            <search_type>is not</search_type>
            <value>%VERSION%</value>
            <opening_paren>false</opening_paren>
            <closing_paren>false</closing_paren>
        </criterion>
        <criterion>
            <name>Computer Group</name>
            <priority>2</priority>
            <and_or>and</and_or>
            <search_type>member of</search_type>
            <value>Testing</value>
            <opening_paren>false</opening_paren>
            <closing_paren>false</closing_paren>
        </criterion>
    </criteria>
</computer_group>
```

SmartGroupTemplate.xml



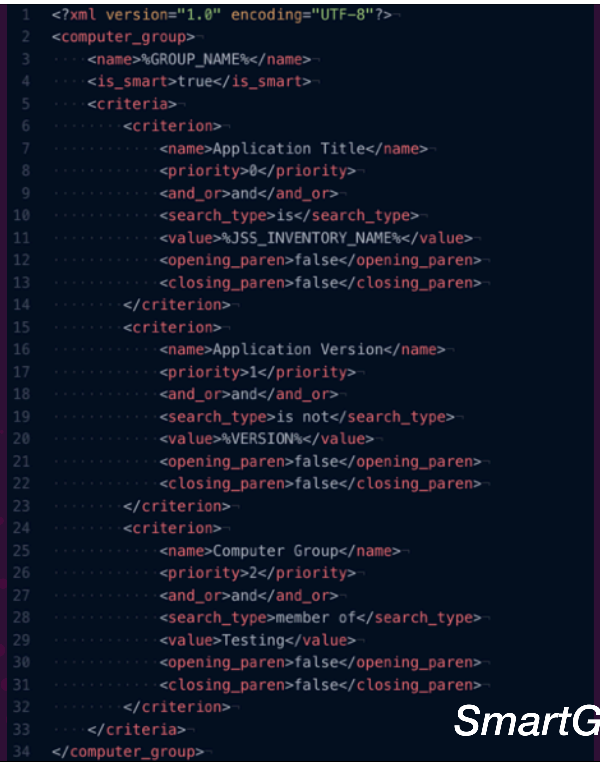
```xml
<?xml version="1.0" encoding="UTF-8"?>
<computer_group>
    <name>%GROUP_NAME%</name>
    <is_smart>true</is_smart>
    <criteria>
        <criterion>
            <name>Application Title</name>
            <priority>0</priority>
            <and_or>and</and_or>
            <search_type>is</search_type>
            <value>%JSS_INVENTORY_NAME%</value>
            <opening_paren>false</opening_paren>
            <closing_paren>false</closing_paren>
        </criterion>
        <criterion>
            <name>Application Version</name>
            <priority>1</priority>
            <and_or>and</and_or>
            <search_type>is not</search_type>
            <value>%VERSION%</value>
            <opening_paren>false</opening_paren>
            <closing_paren>false</closing_paren>
        </criterion>
        <criterion>
            <name>Computer Group</name>
            <priority>2</priority>
            <and_or>and</and_or>
            <search_type>member of</search_type>
            <value>Testing</value>
            <opening_paren>false</opening_paren>
            <closing_paren>false</closing_paren>
        </criterion>
    </criteria>
</computer_group>
```

SmartGroupTemplate.xml



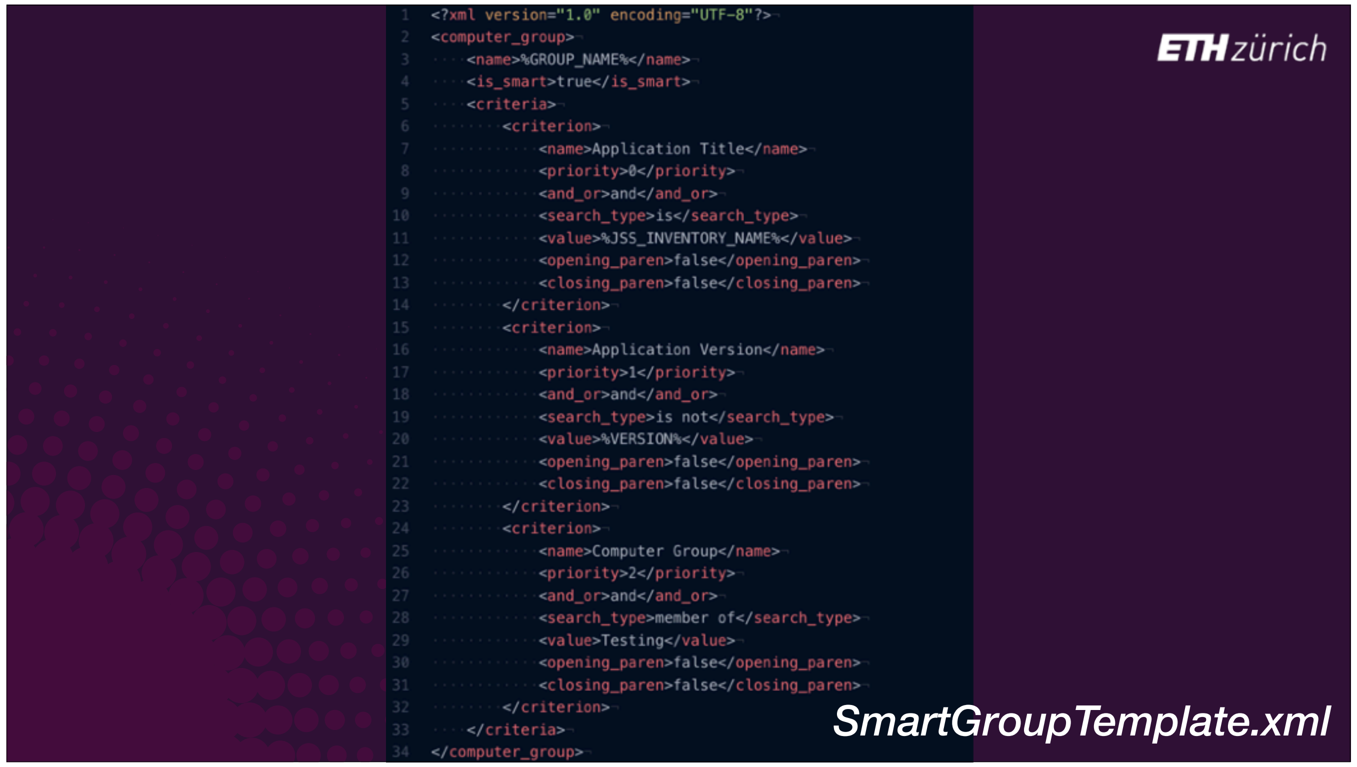
```xml
<?xml version="1.0" encoding="UTF-8"?>
<computer_group>
    <name>%GROUP_NAME%</name>
    <is_smart>true</is_smart>
    <criteria>
        <criterion>
            <name>Application Title</name>
            <priority>0</priority>
            <and_or>and</and_or>
            <search_type>is</search_type>
            <value>%JSS_INVENTORY_NAME%</value>
            <opening_paren>false</opening_paren>
            <closing_paren>false</closing_paren>
        </criterion>
        <criterion>
            <name>Application Version</name>
            <priority>1</priority>
            <and_or>and</and_or>
            <search_type>is not</search_type>
            <value>%VERSION%</value>
            <opening_paren>false</opening_paren>
            <closing_paren>false</closing_paren>
        </criterion>
        <criterion>
            <name>Computer Group</name>
            <priority>2</priority>
            <and_or>and</and_or>
            <search_type>member of</search_type>
            <value>Testing</value>
            <opening_paren>false</opening_paren>
            <closing_paren>false</closing_paren>
        </criterion>
    </criteria>
</computer_group>
```

SmartGroupTemplate.xml



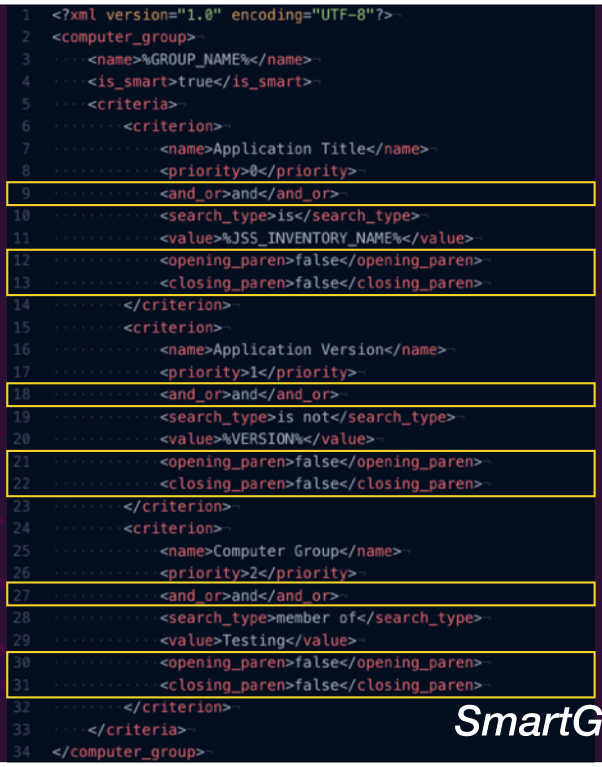
```xml
<?xml version="1.0" encoding="UTF-8"?>
<computer_group>
    <name>%GROUP_NAME%</name>
    <is_smart>true</is_smart>
    <criteria>
        <criterion>
            <name>Application Title</name>
            <priority>0</priority>
            <and_or>and</and_or>
            <search_type>is</search_type>
            <value>%JSS_INVENTORY_NAME%</value>
            <opening_paren>false</opening_paren>
            <closing_paren>false</closing_paren>
        </criterion>
        <criterion>
            <name>Application Version</name>
            <priority>1</priority>
            <and_or>and</and_or>
            <search_type>is not</search_type>
            <value>%VERSION%</value>
            <opening_paren>false</opening_paren>
            <closing_paren>false</closing_paren>
        </criterion>
        <criterion>
            <name>Computer Group</name>
            <priority>2</priority>
            <and_or>and</and_or>
            <search_type>member of</search_type>
            <value>Testing</value>
            <opening_paren>false</opening_paren>
            <closing_paren>false</closing_paren>
        </criterion>
    </criteria>
</computer_group>
```

SmartGroupTemplate.xml



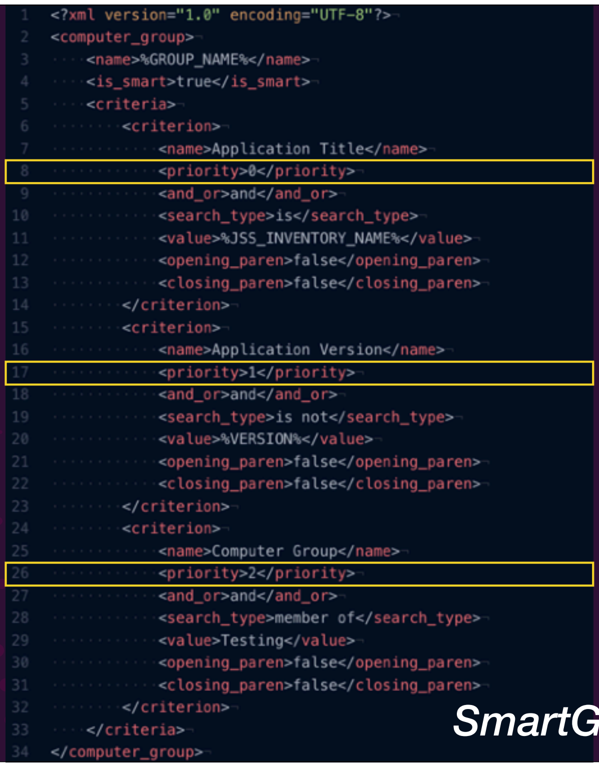
```xml
<?xml version="1.0" encoding="UTF-8"?>
<computer_group>
    <name>%GROUP_NAME%</name>
    <is_smart>true</is_smart>
    <criteria>
        <criterion>
            <name>Application Title</name>
            <priority>0</priority>
            <and_or>and</and_or>
            <search_type>is</search_type>
            <value>%JSS_INVENTORY_NAME%</value>
            <opening_paren>false</opening_paren>
            <closing_paren>false</closing_paren>
        </criterion>
        <criterion>
            <name>Application Version</name>
            <priority>1</priority>
            <and_or>and</and_or>
            <search_type>is not</search_type>
            <value>%VERSION%</value>
            <opening_paren>false</opening_paren>
            <closing_paren>false</closing_paren>
        </criterion>
        <criterion>
            <name>Computer Group</name>
            <priority>2</priority>
            <and_or>and</and_or>
            <search_type>member of</search_type>
            <value>Testing</value>
            <opening_paren>false</opening_paren>
            <closing_paren>false</closing_paren>
        </criterion>
    </criteria>
</computer_group>
```

- If you do make your own group templates, [click] note that there are keys for changing the 'and/or' options, and for setting opening and closing parentheses, just like in the GUI.
- Also, make sure the priority keys for each criterion are in order from 0 upwards, otherwise you can end up with an empty smart group, which is a very bad idea in Jamf...



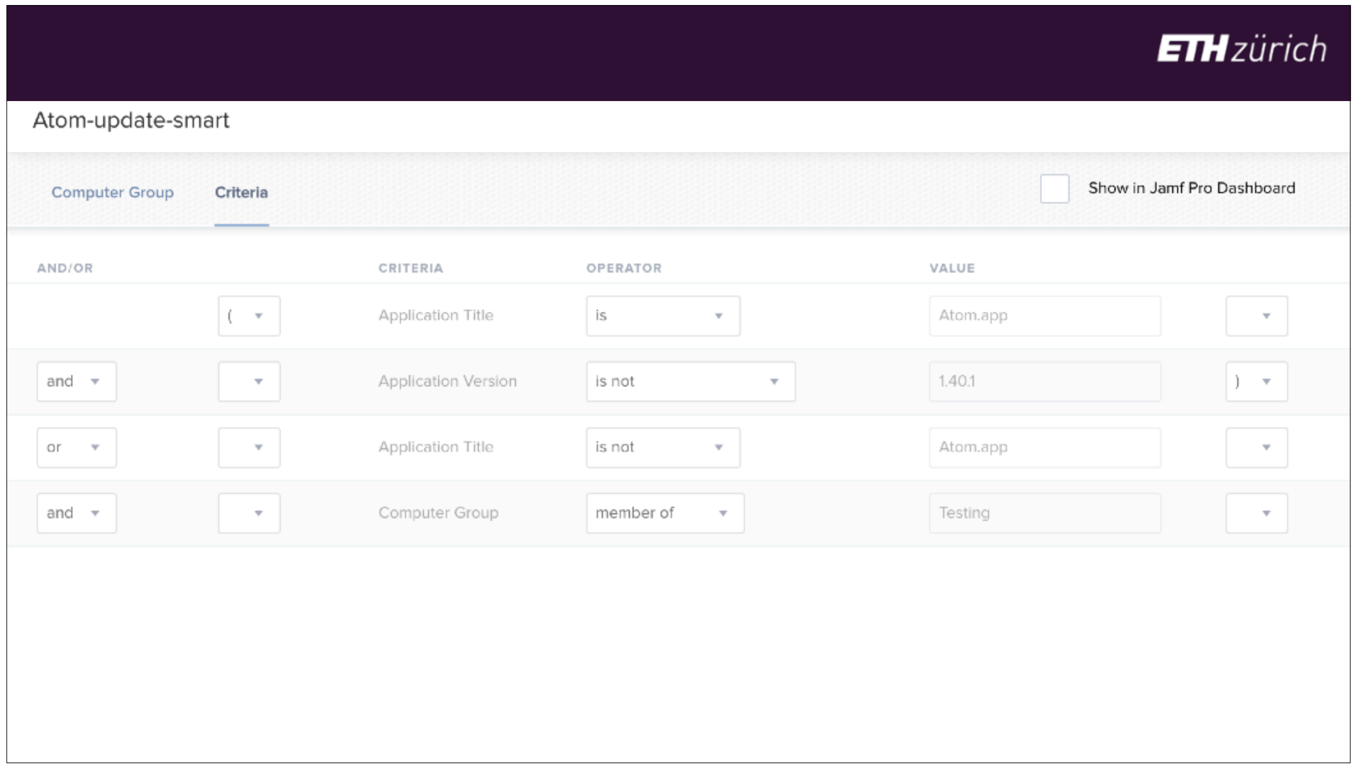
```xml
<?xml version="1.0" encoding="UTF-8"?>
<computer_group>
    <name>%GROUP_NAME%</name>
    <is_smart>true</is_smart>
    <criteria>
        <criterion>
            <name>Application Title</name>
            <priority>0</priority>
            <and_or>and</and_or>
            <search_type>is</search_type>
            <value>%JSS_INVENTORY_NAME%</value>
            <opening_paren>false</opening_paren>
            <closing_paren>false</closing_paren>
        </criterion>
        <criterion>
            <name>Application Version</name>
            <priority>1</priority>
            <and_or>and</and_or>
            <search_type>is not</search_type>
            <value>%VERSION%</value>
            <opening_paren>false</opening_paren>
            <closing_paren>false</closing_paren>
        </criterion>
        <criterion>
            <name>Computer Group</name>
            <priority>2</priority>
            <and_or>and</and_or>
            <search_type>member of</search_type>
            <value>Testing</value>
            <opening_paren>false</opening_paren>
            <closing_paren>false</closing_paren>
        </criterion>
    </criteria>
</computer_group>
```

SmartGroupTemplate.xml



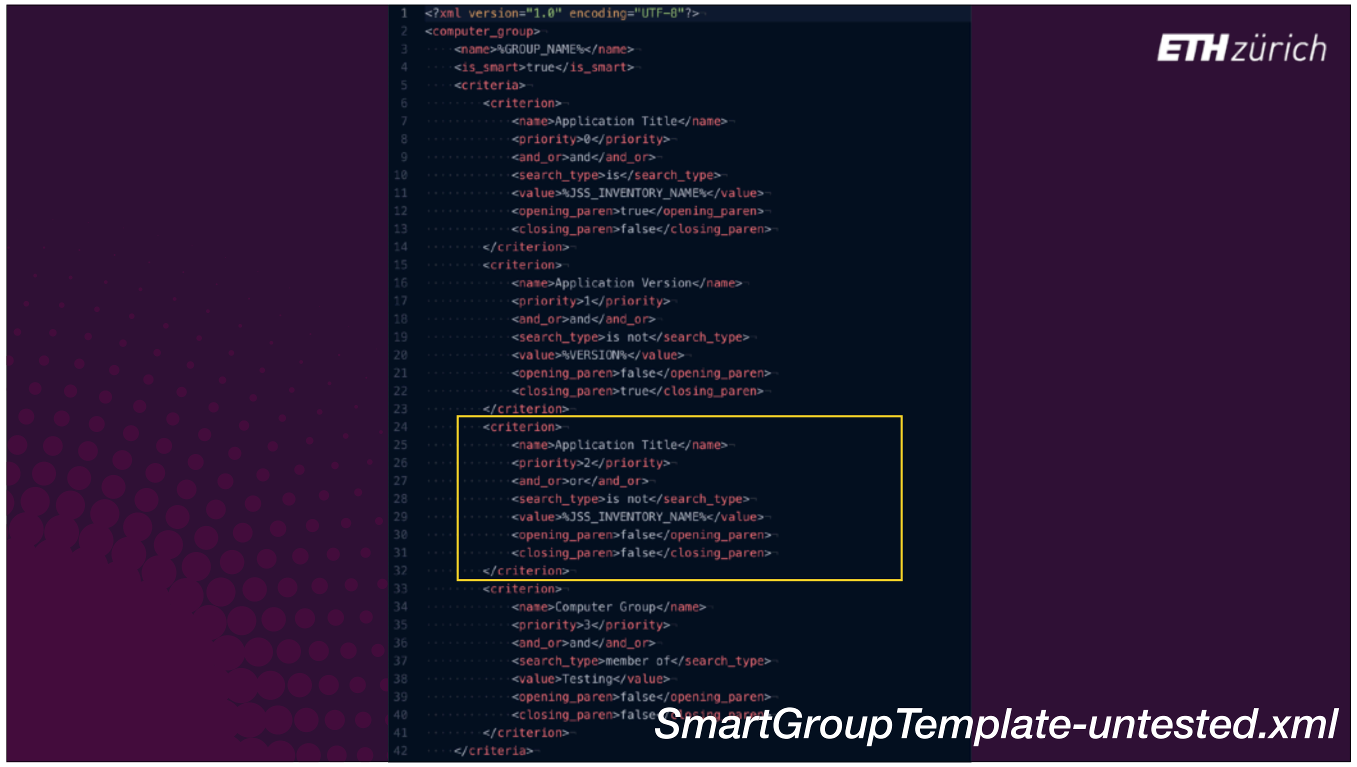
```xml
<?xml version="1.0" encoding="UTF-8"?>
<computer_group>
    <name>%GROUP_NAME%</name>
    <is_smart>true</is_smart>
    <criteria>
        <criterion>
            <name>Application Title</name>
            <priority>0</priority>
            <and_or>and</and_or>
            <search_type>is</search_type>
            <value>%JSS_INVENTORY_NAME%</value>
            <opening_paren>false</opening_paren>
            <closing_paren>false</closing_paren>
        </criterion>
        <criterion>
            <name>Application Version</name>
            <priority>1</priority>
            <and_or>and</and_or>
            <search_type>is not</search_type>
            <value>%VERSION%</value>
            <opening_paren>false</opening_paren>
            <closing_paren>false</closing_paren>
        </criterion>
        <criterion>
            <name>Computer Group</name>
            <priority>2</priority>
            <and_or>and</and_or>
            <search_type>member of</search_type>
            <value>Testing</value>
            <opening_paren>false</opening_paren>
            <closing_paren>false</closing_paren>
        </criterion>
    </criteria>
</computer_group>
```

SmartGroupTemplate.xml



AND/OR		CRITERIA	OPERATOR	VALUE	
	(	Application Title	is	Atom.app	
and		Application Version	is not	1.40.1	)
or		Application Title	is not	Atom.app	
and		Computer Group	member of	Testing	

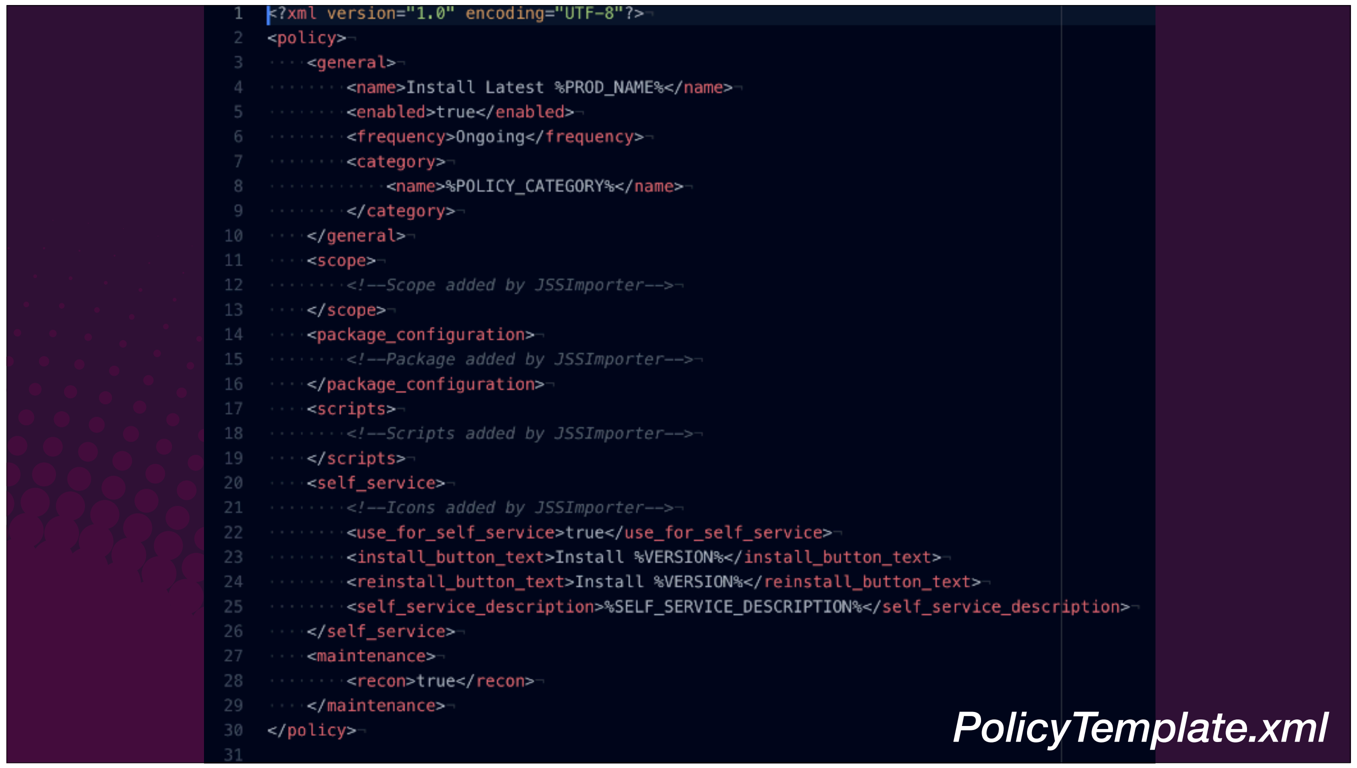
- [slide was not shown in the session]
- I want all my Testing computers to see every app that's available for testing, not just computers with an older version of the untested app already installed.
- so, I want my smart groups to look like this,
- Where either the computer has the app installed but not the version that is provided in the policy,
- OR the computer does not have the app installed at all,
- AND still that it should be in the Testing group.
- To do this we need parentheses around the first two, and change the and/or value to OR.


ETH zürich

```xml
<?xml version="1.0" encoding="UTF-8"?>
<computer_group>
    <name>%GROUP_NAME%</name>
    <is_smart>true</is_smart>
    <criteria>
        <criterion>
            <name>Application Title</name>
            <priority>0</priority>
            <and_or>and</and_or>
            <search_type>is</search_type>
            <value>%JSS_INVENTORY_NAME%</value>
            <opening_paren>true</opening_paren>
            <closing_paren>false</closing_paren>
        </criterion>
        <criterion>
            <name>Application Version</name>
            <priority>1</priority>
            <and_or>and</and_or>
            <search_type>is not</search_type>
            <value>%VERSION%</value>
            <opening_paren>false</opening_paren>
            <closing_paren>true</closing_paren>
        </criterion>
        <criterion>
            <name>Application Title</name>
            <priority>2</priority>
            <and_or>or</and_or>
            <search_type>is not</search_type>
            <value>%JSS_INVENTORY_NAME%</value>
            <opening_paren>false</opening_paren>
            <closing_paren>false</closing_paren>
        </criterion>
        <criterion>
            <name>Computer Group</name>
            <priority>3</priority>
            <and_or>and</and_or>
            <search_type>member of</search_type>
            <value>Testing</value>
            <opening_paren>false</opening_paren>
            <closing_paren>false</closing_paren>
        </criterion>
    </criteria>
```

SmartGroupTemplate-untested.xml

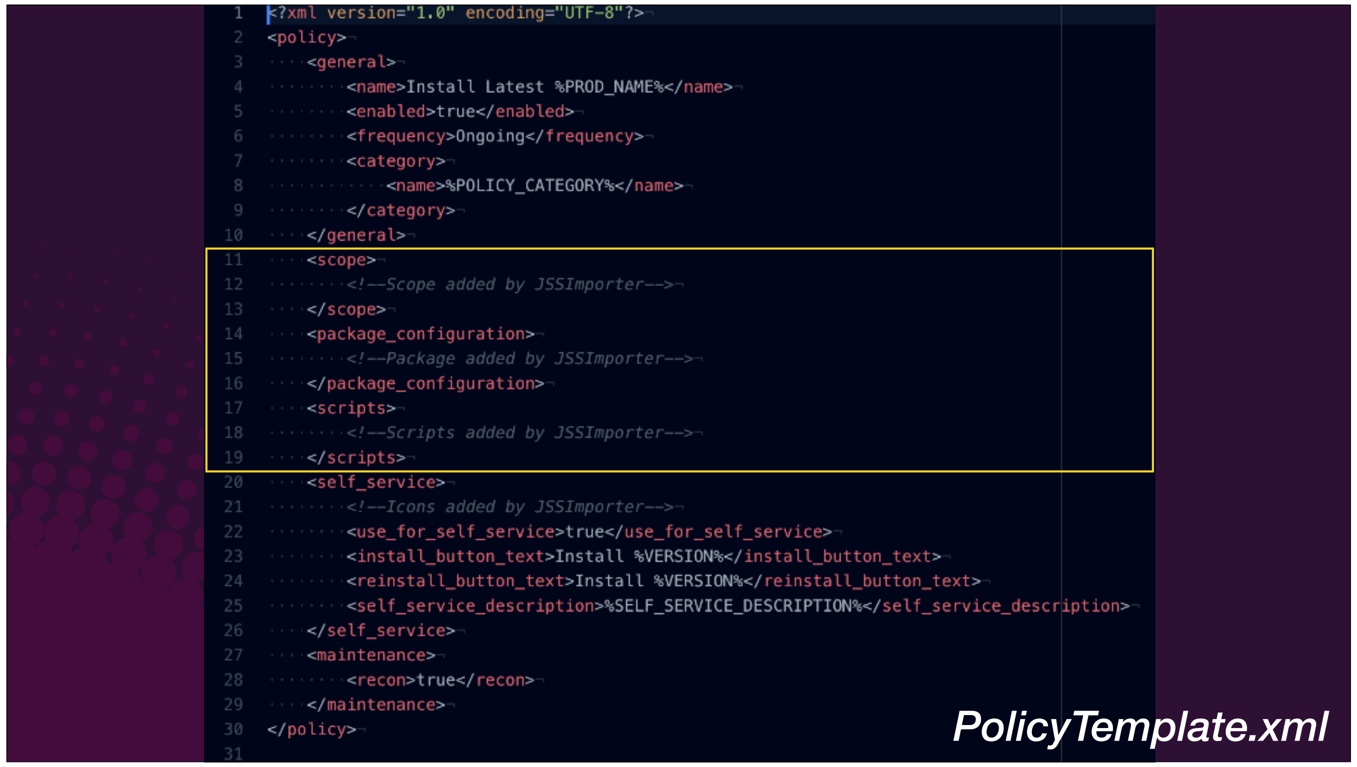
- [slide was not shown in the session]
- In this smart group template,
- [click] I add the criterion that the computer does NOT have the Application Title provided in the JSS recipe.
- [click] I keep the two criteria that state if a different version of the app is installed on the computer, the policy will show in Self Service.
- [click] And I add open and closing parentheses to the Title/Version criteria, and an OR search type to the alternative criteria.



```xml
<?xml version="1.0" encoding="UTF-8"?>
<policy>
    <general>
        <name>Install Latest %PROD_NAME%</name>
        <enabled>true</enabled>
        <frequency>Ongoing</frequency>
        <category>
            <name>%POLICY_CATEGORY%</name>
        </category>
    </general>
    <scope>
        <!--Scope added by JSSImporter-->
    </scope>
    <package_configuration>
        <!--Package added by JSSImporter-->
    </package_configuration>
    <scripts>
        <!--Scripts added by JSSImporter-->
    </scripts>
    <self_service>
        <!--Icons added by JSSImporter-->
        <use_for_self_service>true</use_for_self_service>
        <install_button_text>Install %VERSION%</install_button_text>
        <reinstall_button_text>Install %VERSION%</reinstall_button_text>
        <self_service_description>%SELF_SERVICE_DESCRIPTION%</self_service_description>
    </self_service>
    <maintenance>
        <recon>true</recon>
    </maintenance>
</policy>
```

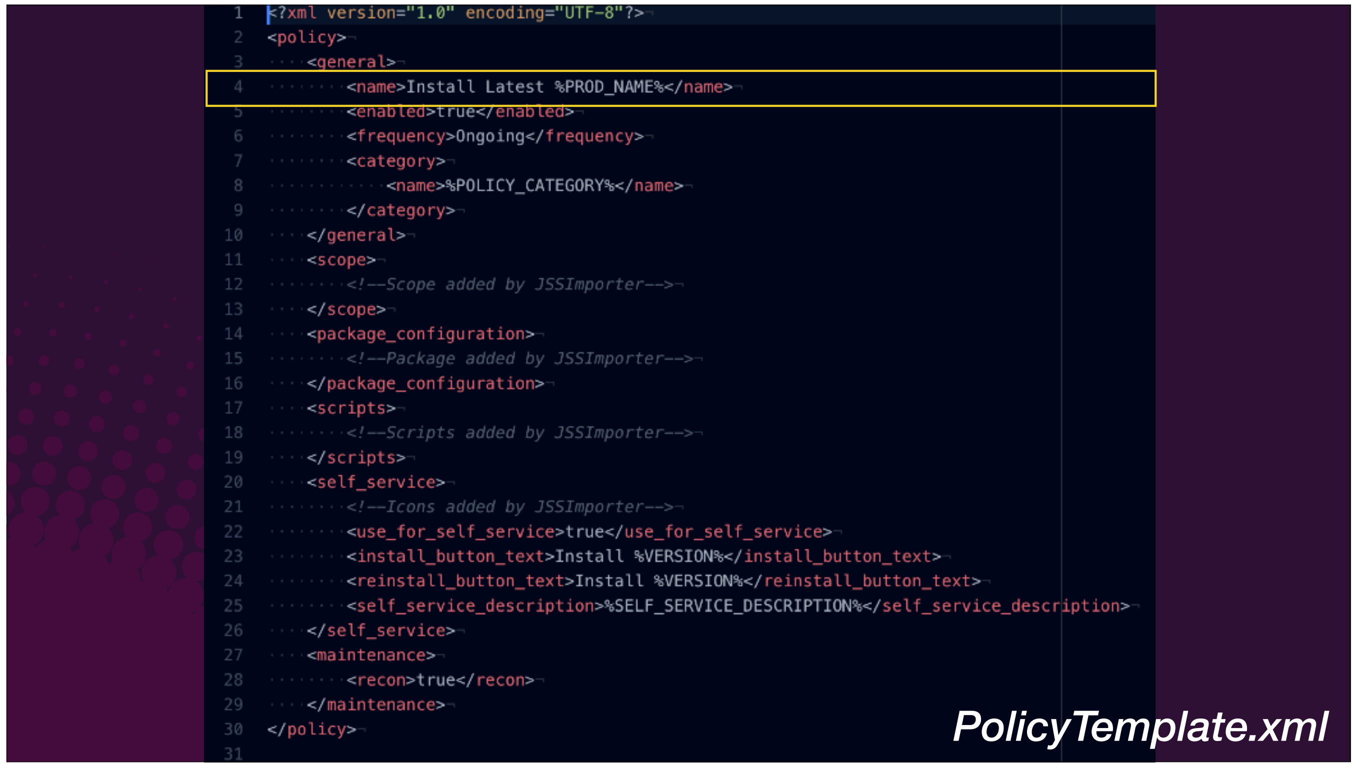
PolicyTemplate.xml

- Let's also look at the standard policy template to see what we can override in there.
- [click] First notice the scope, package config and scripts are entirely handled by JSSImporter, so you don't usually edit these in the template as you can set them as you wish in the recipe override.
- Note that the policy name is not completely overridable -'Install Latest' is hard-coded into the template.
- [click to move] the same is true of the Self Service button text, where the word 'Install' is hard-coded in.
- If you want to be able to change these values more fundamentally, you have to edit the standard policy template, so you have to make a copy in your Recipe Overrides folder.



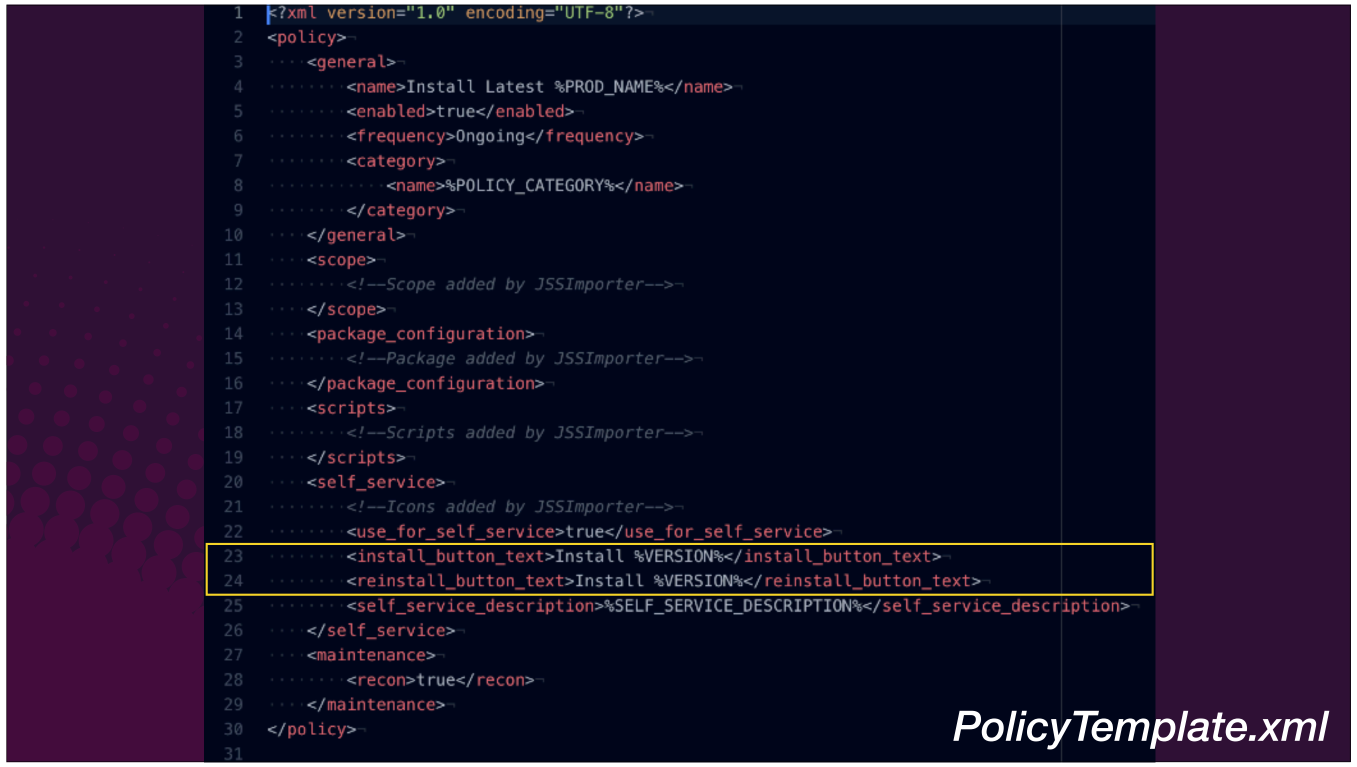
```
<?xml version="1.0" encoding="UTF-8"?>
<policy>
    <general>
        <name>Install Latest %PROD_NAME%</name>
        <enabled>true</enabled>
        <frequency>Ongoing</frequency>
        <category>
            <name>%POLICY_CATEGORY%</name>
        </category>
    </general>
    <scope>
        <!--Scope added by JSSImporter-->
    </scope>
    <package_configuration>
        <!--Package added by JSSImporter-->
    </package_configuration>
    <scripts>
        <!--Scripts added by JSSImporter-->
    </scripts>
    <self_service>
        <!--Icons added by JSSImporter-->
        <use_for_self_service>true</use_for_self_service>
        <install_button_text>Install %VERSION%</install_button_text>
        <reinstall_button_text>Install %VERSION%</reinstall_button_text>
        <self_service_description>%SELF_SERVICE_DESCRIPTION%</self_service_description>
    </self_service>
    <maintenance>
        <recon>true</recon>
    </maintenance>
</policy>
```

PolicyTemplate.xml



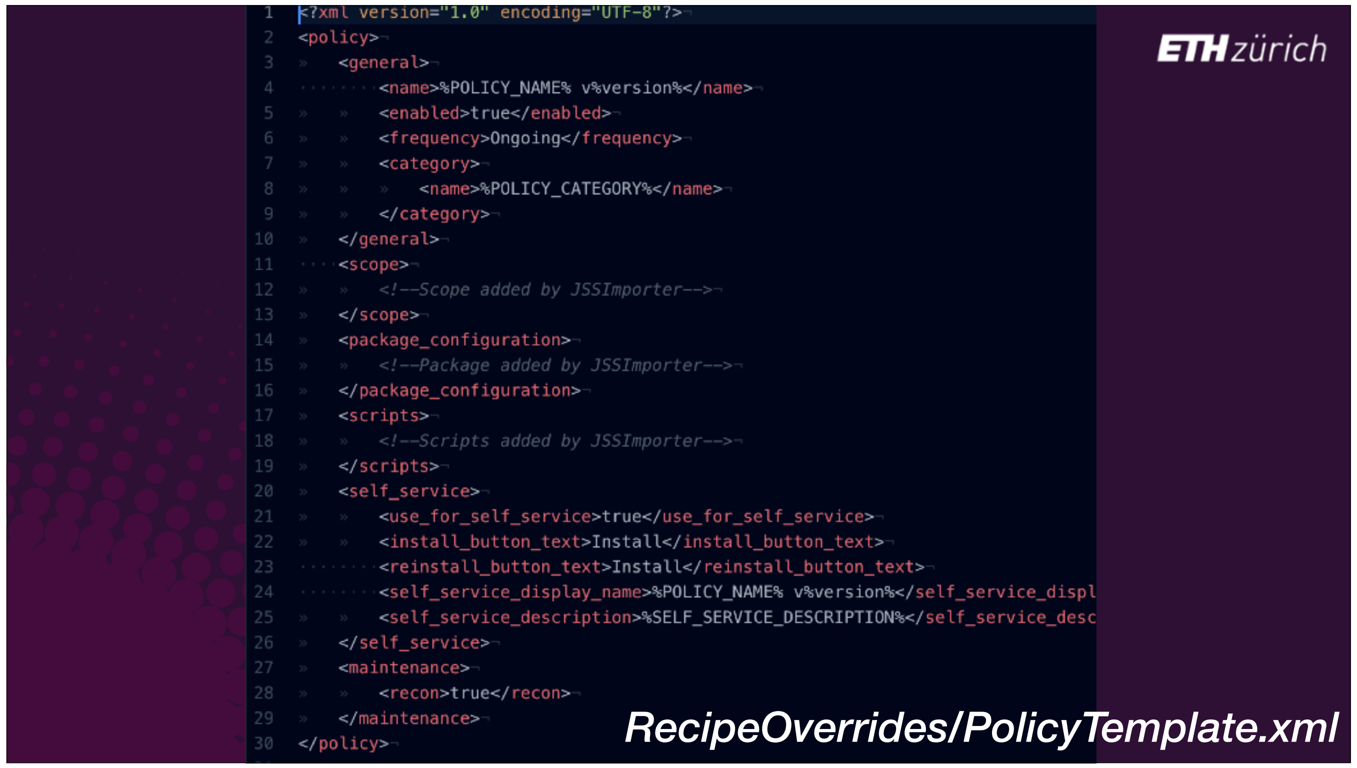
```
<?xml version="1.0" encoding="UTF-8"?>
<policy>
    <general>
        <name>Install Latest %PROD_NAME%</name>
        <enabled>true</enabled>
        <frequency>Ongoing</frequency>
        <category>
            <name>%POLICY_CATEGORY%</name>
        </category>
    </general>
    <scope>
        <!--Scope added by JSSImporter-->
    </scope>
    <package_configuration>
        <!--Package added by JSSImporter-->
    </package_configuration>
    <scripts>
        <!--Scripts added by JSSImporter-->
    </scripts>
    <self_service>
        <!--Icons added by JSSImporter-->
        <use_for_self_service>true</use_for_self_service>
        <install_button_text>Install %VERSION%</install_button_text>
        <reinstall_button_text>Install %VERSION%</reinstall_button_text>
        <self_service_description>%SELF_SERVICE_DESCRIPTION%</self_service_description>
    </self_service>
    <maintenance>
        <recon>true</recon>
    </maintenance>
</policy>
```

PolicyTemplate.xml



```xml
<?xml version="1.0" encoding="UTF-8"?>
<policy>
    <general>
        <name>Install Latest %PROD_NAME%</name>
        <enabled>true</enabled>
        <frequency>Ongoing</frequency>
        <category>
            <name>%POLICY_CATEGORY%</name>
        </category>
    </general>
    <scope>
        <!--Scope added by JSSImporter-->
    </scope>
    <package_configuration>
        <!--Package added by JSSImporter-->
    </package_configuration>
    <scripts>
        <!--Scripts added by JSSImporter-->
    </scripts>
    <self_service>
        <!--Icons added by JSSImporter-->
        <use_for_self_service>true</use_for_self_service>
        <install_button_text>Install %VERSION%</install_button_text>
        <reinstall_button_text>Install %VERSION%</reinstall_button_text>
        <self_service_description>%SELF_SERVICE_DESCRIPTION%</self_service_description>
    </self_service>
    <maintenance>
        <recon>true</recon>
    </maintenance>
</policy>
```

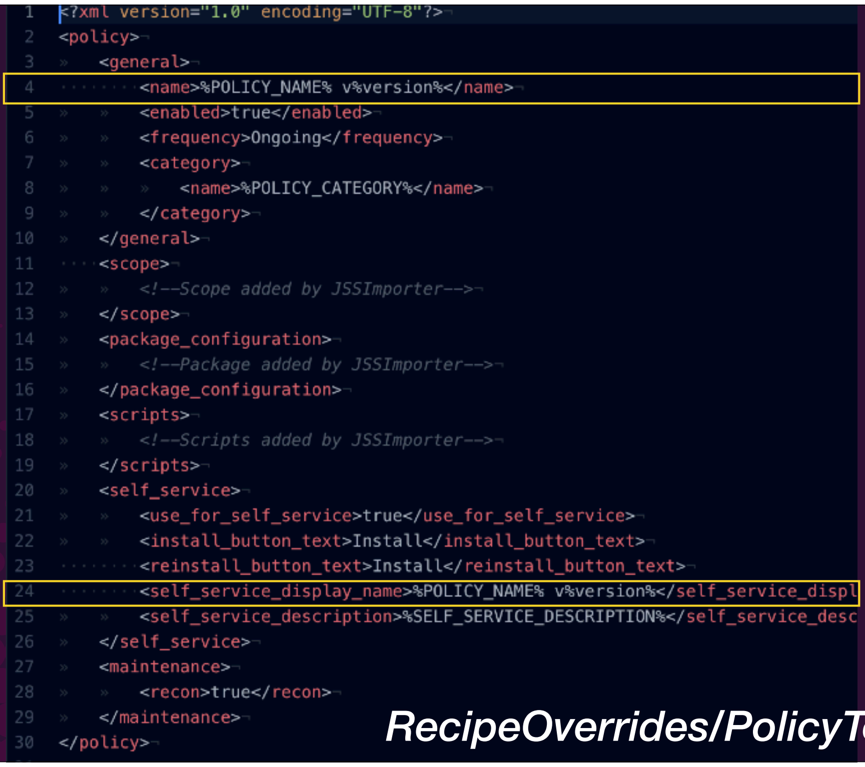
PolicyTemplate.xml



RecipeOverrides/PolicyTemplate.xml

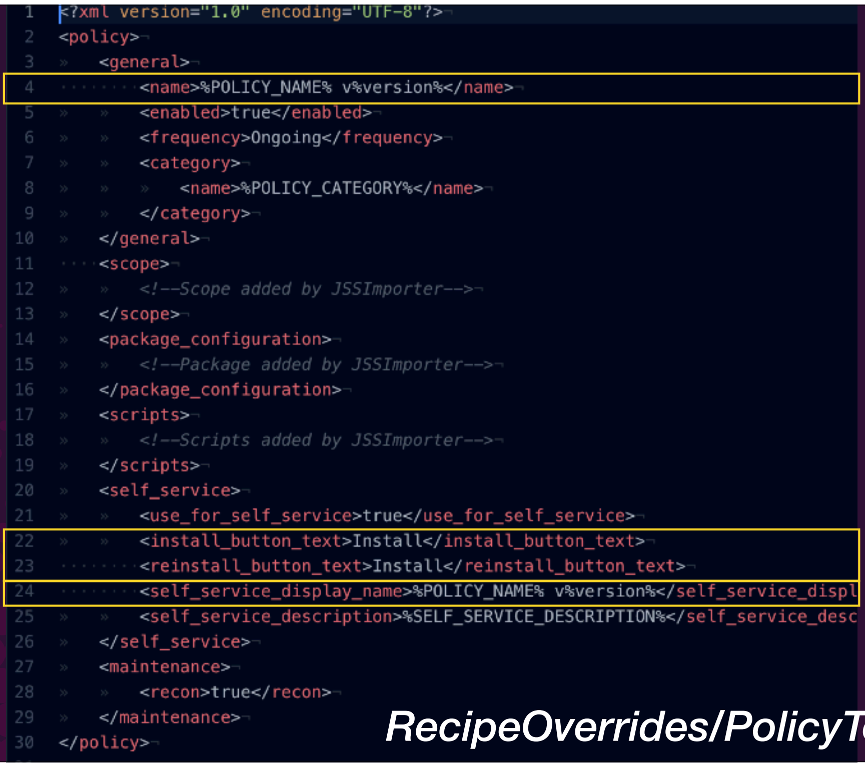
```xml
<?xml version="1.0" encoding="UTF-8"?>
<policy>
    <general>
        <name>%POLICY_NAME% v%version%</name>
        <enabled>true</enabled>
        <frequency>Ongoing</frequency>
        <category>
            <name>%POLICY_CATEGORY%</name>
        </category>
    </general>
    <scope>
        <!--Scope added by JSSImporter-->
    </scope>
    <package_configuration>
        <!--Package added by JSSImporter-->
    </package_configuration>
    <scripts>
        <!--Scripts added by JSSImporter-->
    </scripts>
    <self_service>
        <use_for_self_service>true</use_for_self_service>
        <install_button_text>Install</install_button_text>
        <reinstall_button_text>Install</reinstall_button_text>
        <self_service_display_name>%POLICY_NAME% v%version%</self_service_displ
        <self_service_description>%SELF_SERVICE_DESCRIPTION%</self_service_desc
    </self_service>
    <maintenance>
        <recon>true</recon>
    </maintenance>
</policy>
```

- As an example, here's our main policy template I use at my work.
- [click] We change the policy name and self service name to include the version number.
- [click] we remove the version from the install button.



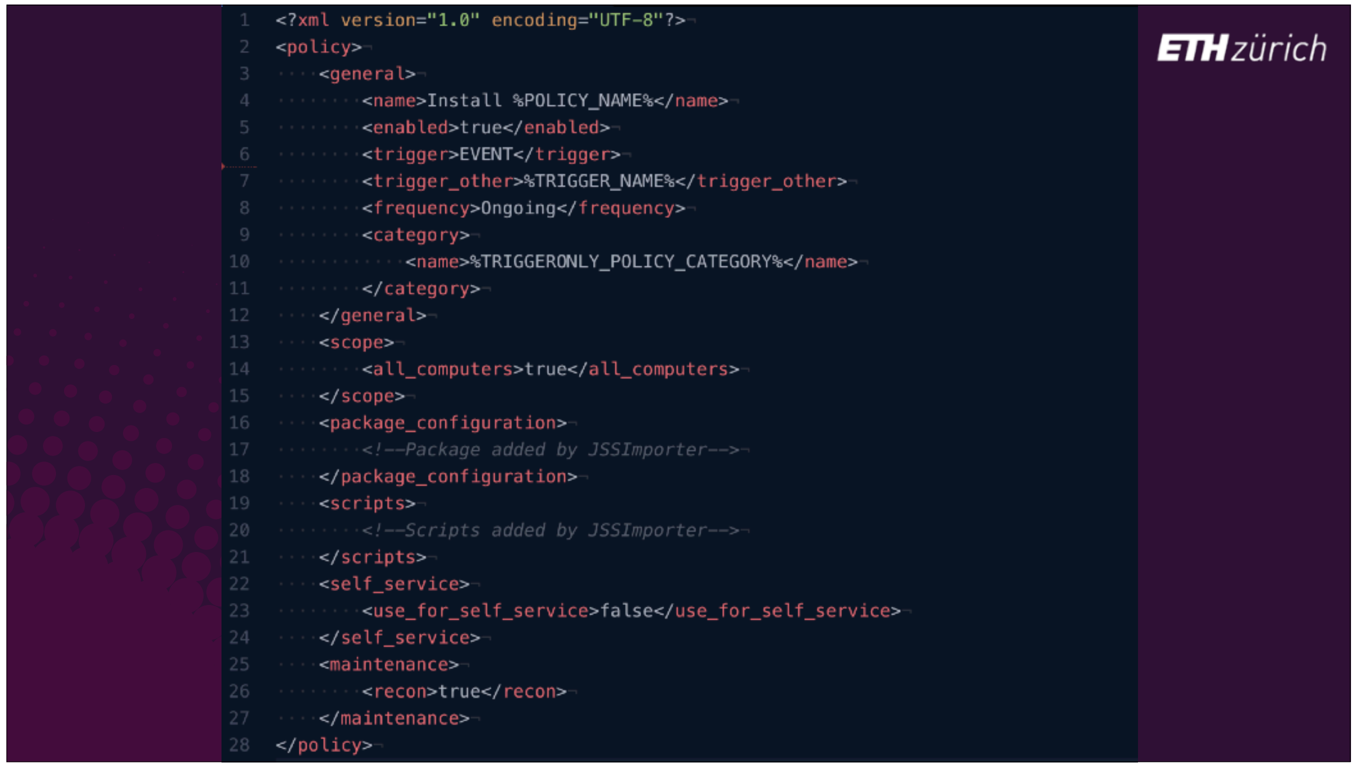
```
<?xml version="1.0" encoding="UTF-8"?>
<policy>
    <general>
        <name>%POLICY_NAME% v%version%</name>
        <enabled>true</enabled>
        <frequency>Ongoing</frequency>
        <category>
            <name>%POLICY_CATEGORY%</name>
        </category>
    </general>
    <scope>
        <!--Scope added by JSSImporter-->
    </scope>
    <package_configuration>
        <!--Package added by JSSImporter-->
    </package_configuration>
    <scripts>
        <!--Scripts added by JSSImporter-->
    </scripts>
    <self_service>
        <use_for_self_service>true</use_for_self_service>
        <install_button_text>Install</install_button_text>
        <reinstall_button_text>Install</reinstall_button_text>
        <self_service_display_name>%POLICY_NAME% v%version%</self_service_displ
        <self_service_description>%SELF_SERVICE_DESCRIPTION%</self_service_desc
    </self_service>
    <maintenance>
        <recon>true</recon>
    </maintenance>
</policy>
```

RecipeOverrides/PolicyTemplate.xml



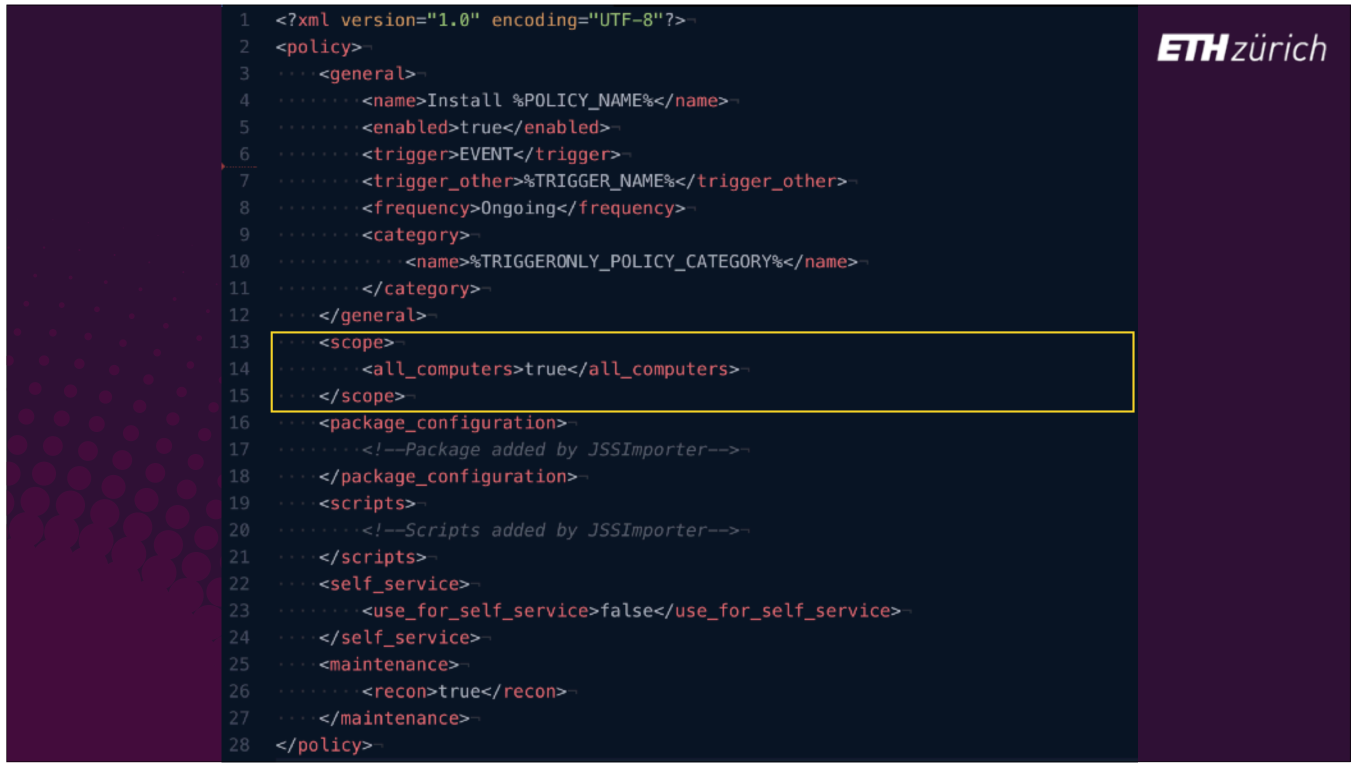
```xml
<?xml version="1.0" encoding="UTF-8"?>
<policy>
    <general>
        <name>%POLICY_NAME% v%version%</name>
        <enabled>true</enabled>
        <frequency>Ongoing</frequency>
        <category>
            <name>%POLICY_CATEGORY%</name>
        </category>
    </general>
    <scope>
        <!--Scope added by JSSImporter-->
    </scope>
    <package_configuration>
        <!--Package added by JSSImporter-->
    </package_configuration>
    <scripts>
        <!--Scripts added by JSSImporter-->
    </scripts>
    <self_service>
        <use_for_self_service>true</use_for_self_service>
        <install_button_text>Install</install_button_text>
        <reinstall_button_text>Install</reinstall_button_text>
        <self_service_display_name>%POLICY_NAME% v%version%</self_service_displ
        <self_service_description>%SELF_SERVICE_DESCRIPTION%</self_service_desc
    </self_service>
    <maintenance>
        <recon>true</recon>
    </maintenance>
</policy>
```

RecipeOverrides/PolicyTemplate.xml

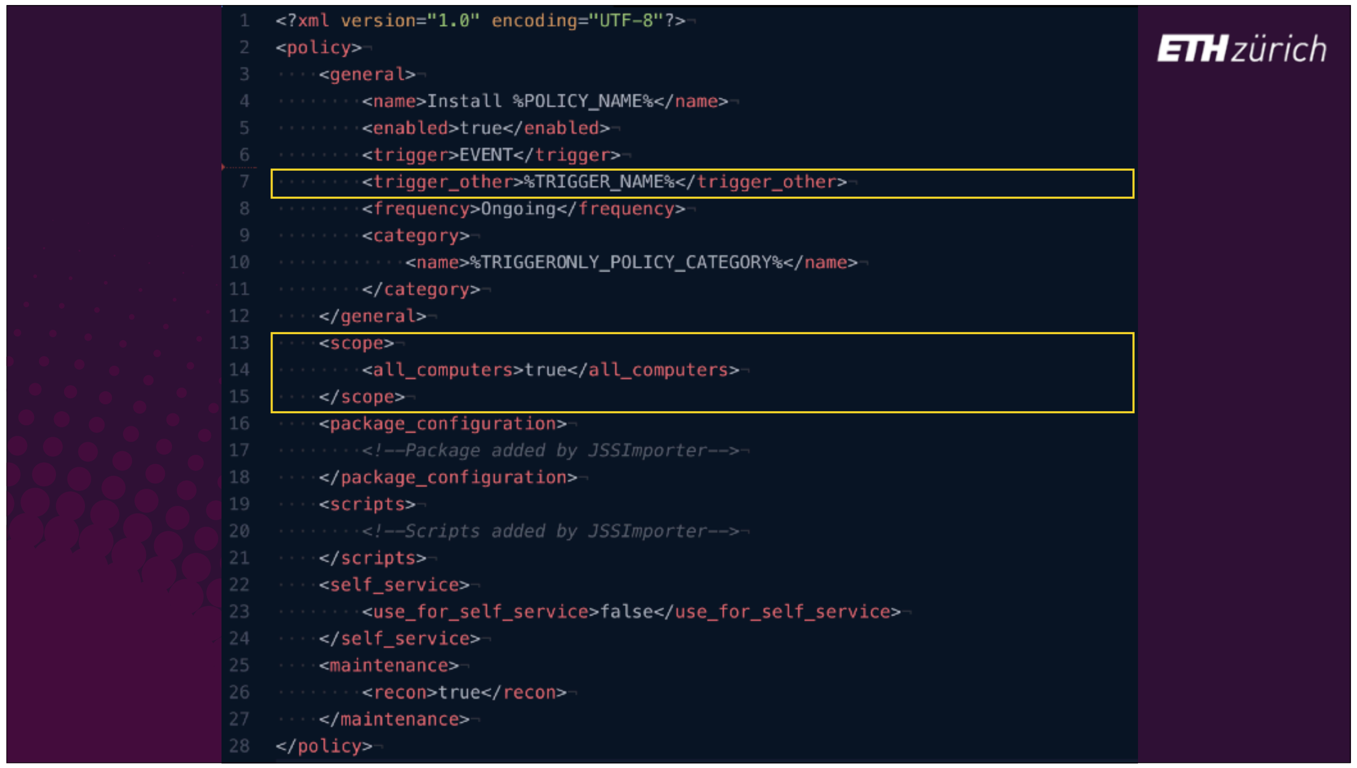


```
<?xml version="1.0" encoding="UTF-8"?>
<policy>
    <general>
        <name>Install %POLICY_NAME%</name>
        <enabled>true</enabled>
        <trigger>EVENT</trigger>
        <trigger_other>%TRIGGER_NAME%</trigger_other>
        <frequency>Ongoing</frequency>
        <category>
            <name>%TRIGGERONLY_POLICY_CATEGORY%</name>
        </category>
    </general>
    <scope>
        <all_computers>true</all_computers>
    </scope>
    <package_configuration>
        <!--Package added by JSSImporter-->
    </package_configuration>
    <scripts>
        <!--Scripts added by JSSImporter-->
    </scripts>
    <self_service>
        <use_for_self_service>false</use_for_self_service>
    </self_service>
    <maintenance>
        <recon>true</recon>
    </maintenance>
</policy>
```

- Here's another example, which is a trigger only policy template.
- This can be used for creating policies that you call from a single enrolment script, for example.
- Instead of using Self Service, this creates a policy that is triggered by an event trigger.
- [click] In this case you can override the scope to All Computers.
- [click] You add the trigger-other key, and can provide the value to that in your recipe override.

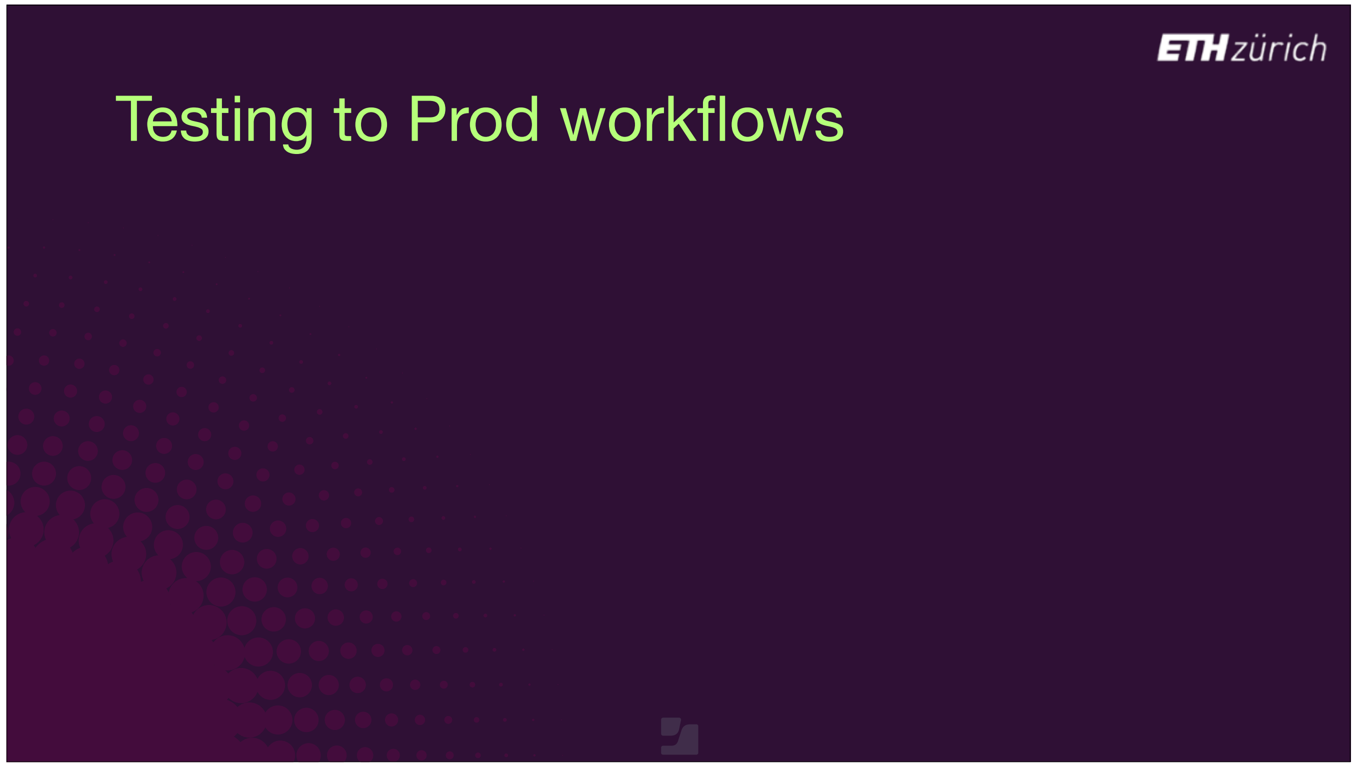


```xml
<?xml version="1.0" encoding="UTF-8"?>
<policy>
    <general>
        <name>Install %POLICY_NAME%</name>
        <enabled>true</enabled>
        <trigger>EVENT</trigger>
        <trigger_other>%TRIGGER_NAME%</trigger_other>
        <frequency>Ongoing</frequency>
        <category>
            <name>%TRIGGERONLY_POLICY_CATEGORY%</name>
        </category>
    </general>
    <scope>
        <all_computers>true</all_computers>
    </scope>
    <package_configuration>
        <!--Package added by JSSImporter-->
    </package_configuration>
    <scripts>
        <!--Scripts added by JSSImporter-->
    </scripts>
    <self_service>
        <use_for_self_service>false</use_for_self_service>
    </self_service>
    <maintenance>
        <recon>true</recon>
    </maintenance>
</policy>
```



ETH zürich

```
<?xml version="1.0" encoding="UTF-8"?>
<policy>
    <general>
        <name>Install %POLICY_NAME%</name>
        <enabled>true</enabled>
        <trigger>EVENT</trigger>
        <trigger_other>%TRIGGER_NAME%</trigger_other>
        <frequency>Ongoing</frequency>
        <category>
            <name>%TRIGGERONLY_POLICY_CATEGORY%</name>
        </category>
    </general>
    <scope>
        <all_computers>true</all_computers>
    </scope>
    <package_configuration>
        <!--Package added by JSSImporter-->
    </package_configuration>
    <scripts>
        <!--Scripts added by JSSImporter-->
    </scripts>
    <self_service>
        <use_for_self_service>false</use_for_self_service>
    </self_service>
    <maintenance>
        <recon>true</recon>
    </maintenance>
</policy>
```



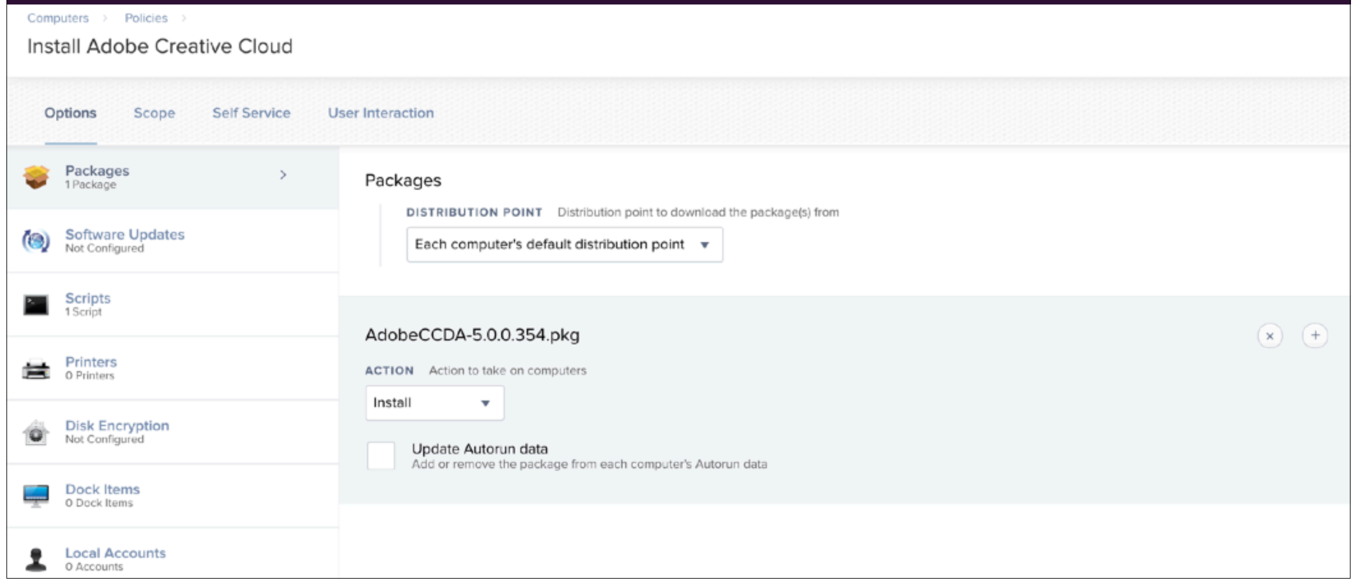
Testing to Prod workflows

- [slide was not shown in the session]

A note on promoting packages to production.
Standard JSS recipes are designed to upload packages into Jamf for testing. Once tested, we then want to deploy those packages in production.

- There's no recipes for this, so most people make their prod policies manually in the Jamf GUI.
- When you update the version that you want to go to production, you just go into your existing production policy and switch the package.
- If you're scoping dynamically, you will also need to go into the smart group and change the current version.

Testing to Prod workflows



Computers › Policies ›

Install Adobe Creative Cloud

Packages

DISTRIBUTION POINT Distribution point to download the package(s) from

Each computer's default distribution point

AdobeCCDA-5.0.0.354.pkg

ACTION Action to take on computers

Install

Update Autorun data
Add or remove the package from each computer's Autorun data

ETH zürich

jss_helper

- [slide was not shown in the session]
- The promotion to production task is made easier using JSS_Helper.
- [click] This script was written by Shea Craig, same as JSSImporter.
- [click] Run 'jss_helper promote', and it identifies policies where there is a newer package available in Jamf compared to the one currently attached to the policy.
- [click] You choose one, and it shows you the newer package or packages.
- In this case there's just one, so just press enter, and your prod policy is automatically updated with the latest package, using the Jamf Pro API.

ETH zürich

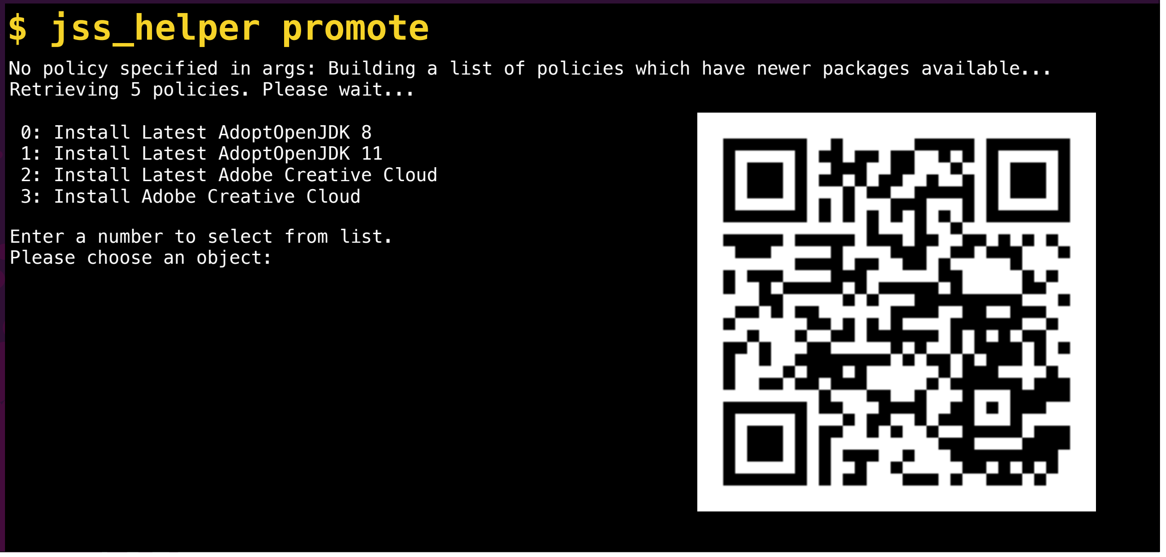
jss_helper

ETH zürich

jss_helper

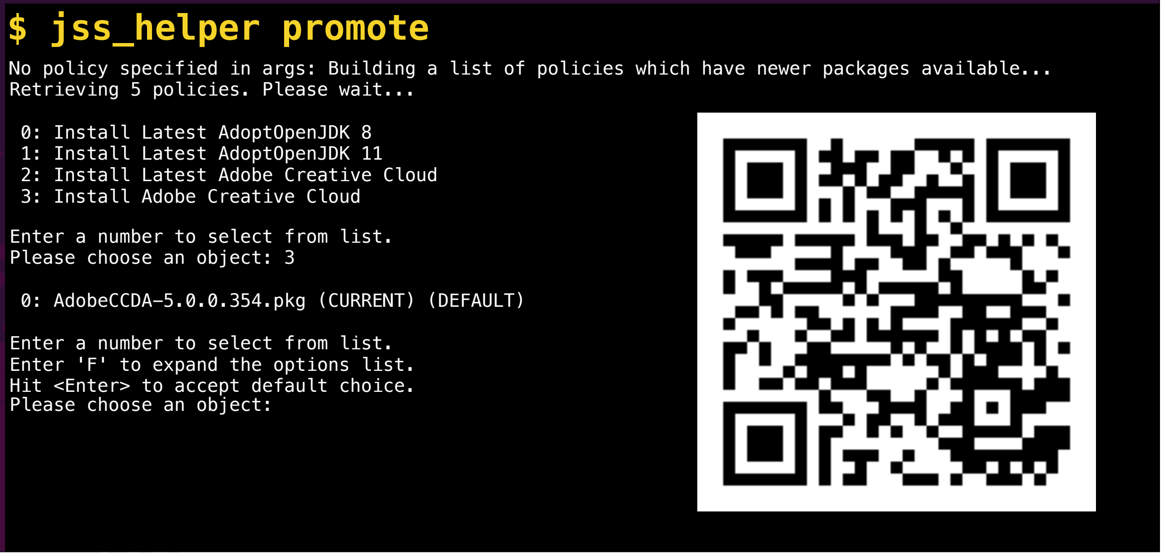
```
$ jss_helper promote
```

jss_helper

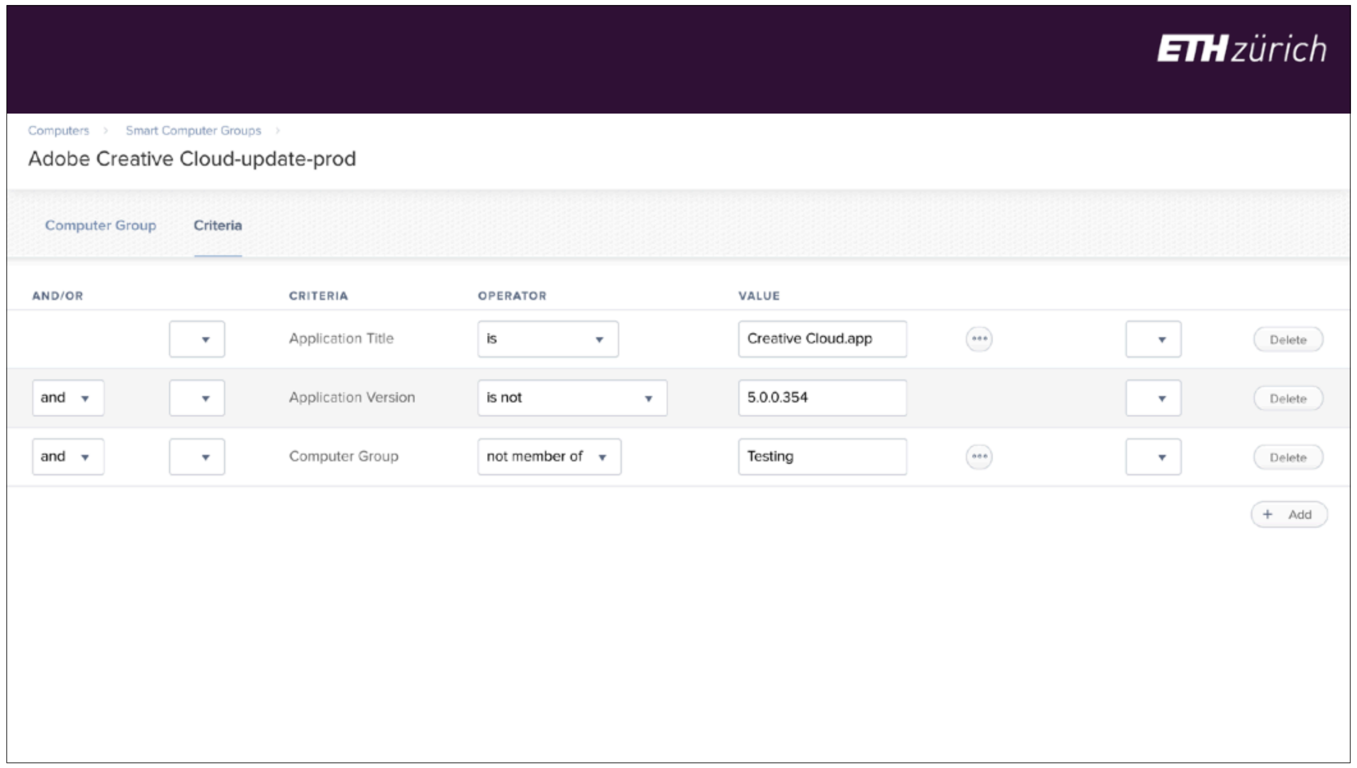


```
$ jss_helper promote
No policy specified in args: Building a list of policies which have newer packages available...
Retrieving 5 policies. Please wait...

 0: Install Latest AdoptOpenJDK 8
 1: Install Latest AdoptOpenJDK 11
 2: Install Latest Adobe Creative Cloud
 3: Install Adobe Creative Cloud

Enter a number to select from list.
Please choose an object:
```

jss_helper



```
$ jss_helper promote
No policy specified in args: Building a list of policies which have newer packages available...
Retrieving 5 policies. Please wait...

 0: Install Latest AdoptOpenJDK 8
 1: Install Latest AdoptOpenJDK 11
 2: Install Latest Adobe Creative Cloud
 3: Install Adobe Creative Cloud

Enter a number to select from list.
Please choose an object: 3

 0: AdobeCCDA-5.0.0.354.pkg (CURRENT) (DEFAULT)

Enter a number to select from list.
Enter 'F' to expand the options list.
Hit <Enter> to accept default choice.
Please choose an object:
```



Computers › Smart Computer Groups ›

Adobe Creative Cloud-update-prod

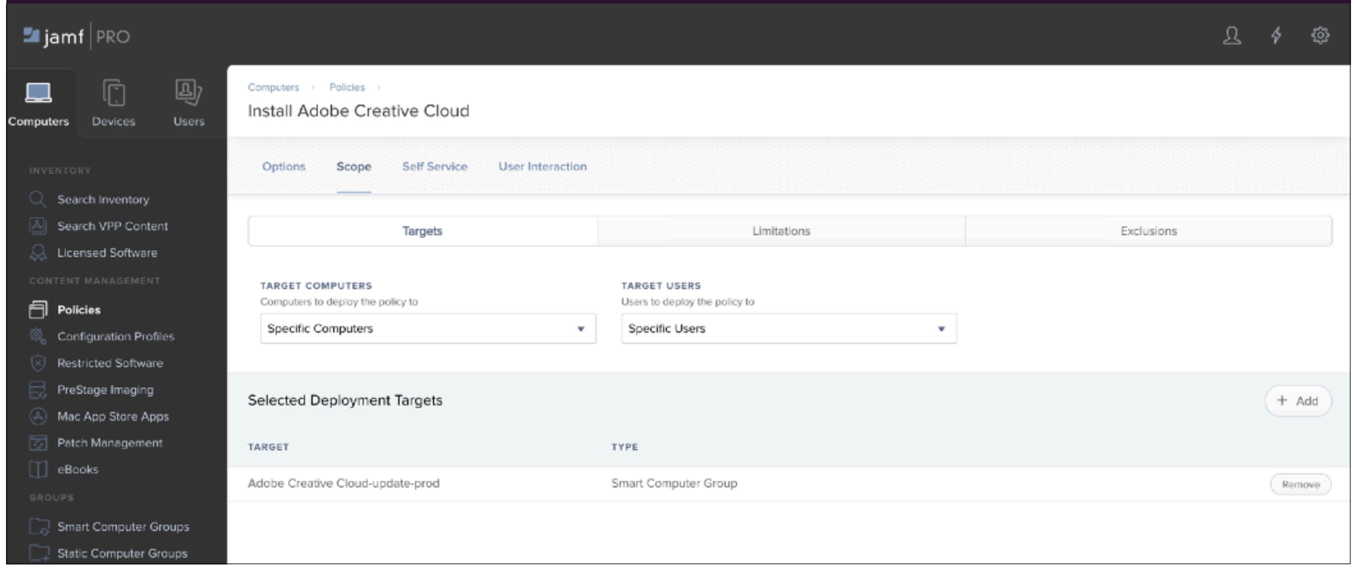
Computer Group | Criteria

AND/OR		CRITERIA	OPERATOR	VALUE
		Application Title	is	Creative Cloud.app
and		Application Version	is not	5.0.0.354
and		Computer Group	not member of	Testing

- [slide was not shown in the session]

If you want to scope your production policies smartly like your testing policies, you'll also need switch the version number in the production smart group.

Testing to Prod workflows



Install Adobe Creative Cloud

TARGET COMPUTERS
Computers to deploy the policy to
Specific Computers

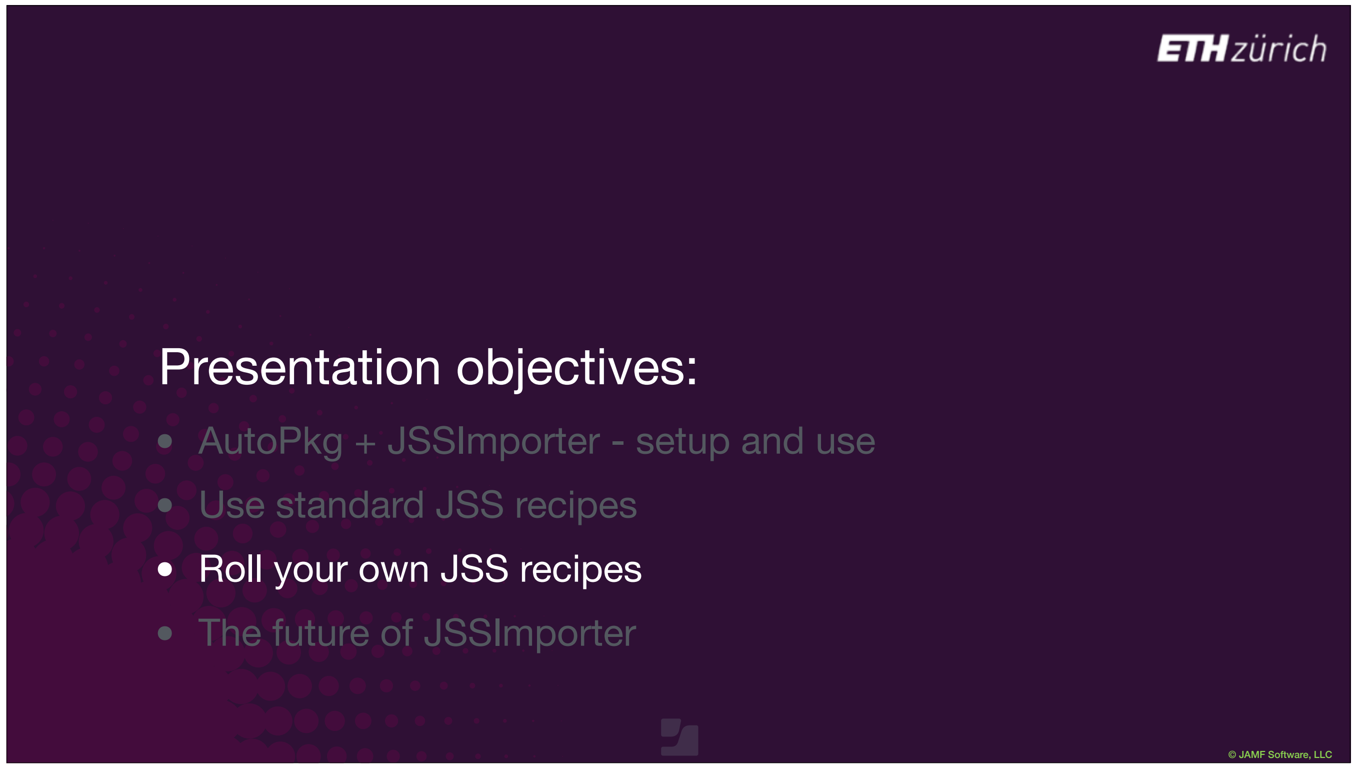
TARGET USERS
Users to deploy the policy to
Specific Users

Selected Deployment Targets

TARGET	TYPE
Adobe Creative Cloud-update-prod	Smart Computer Group

- [slide was not shown in the session]

If you want smart scoping however, you can create a policy with ongoing recurrence, scoped to a smart group similar to the one created by JSSImporter.

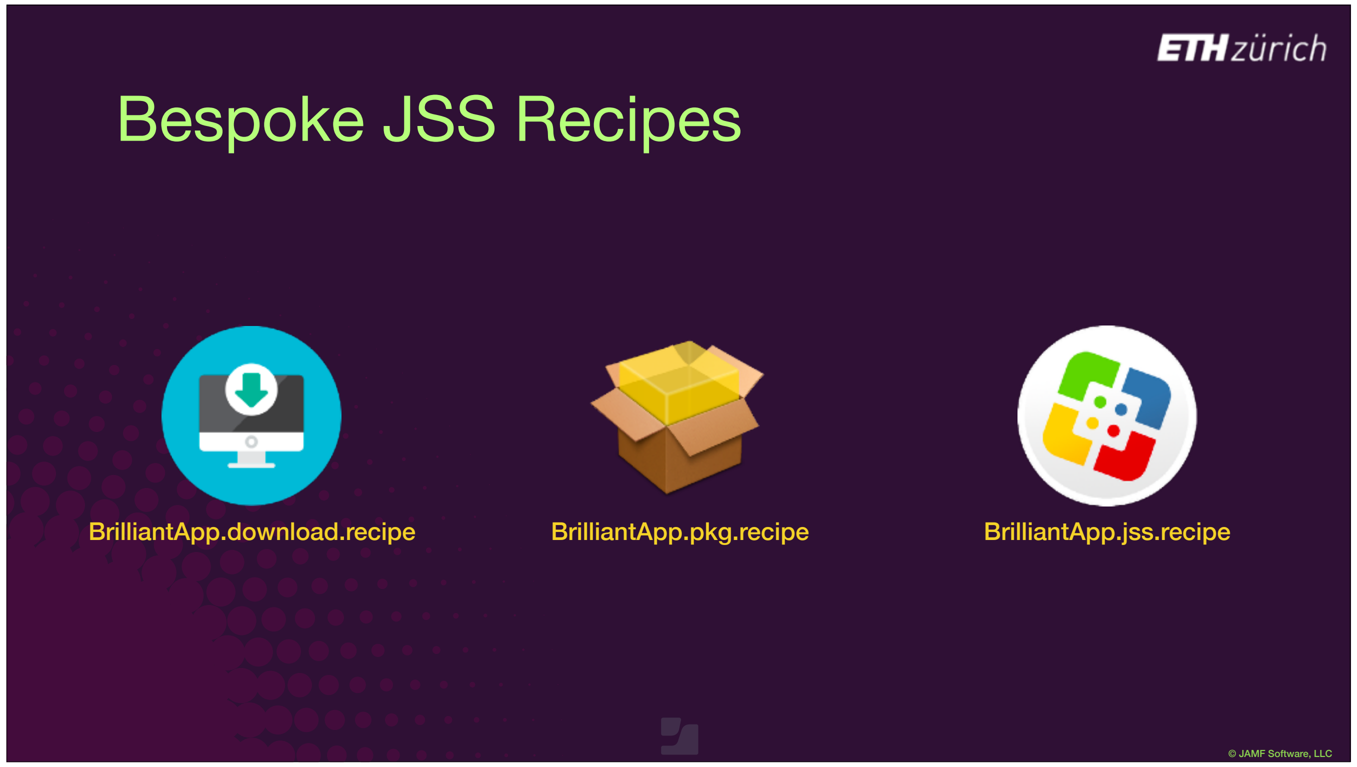


ETH zürich

Presentation objectives:

- AutoPkg + JSSImporter - setup and use
- Use standard JSS recipes
- Roll your own JSS recipes
- The future of JSSImporter

So that was a look at standard recipes, and the extent to which you can customise by overriding variables and templates.
If your needs are not met by the standards, you can still use JSSImporter.
But you may have to create your own recipes.



Bespoke JSS Recipes

BrilliantApp.download.recipe

BrilliantApp.pkg.recipe

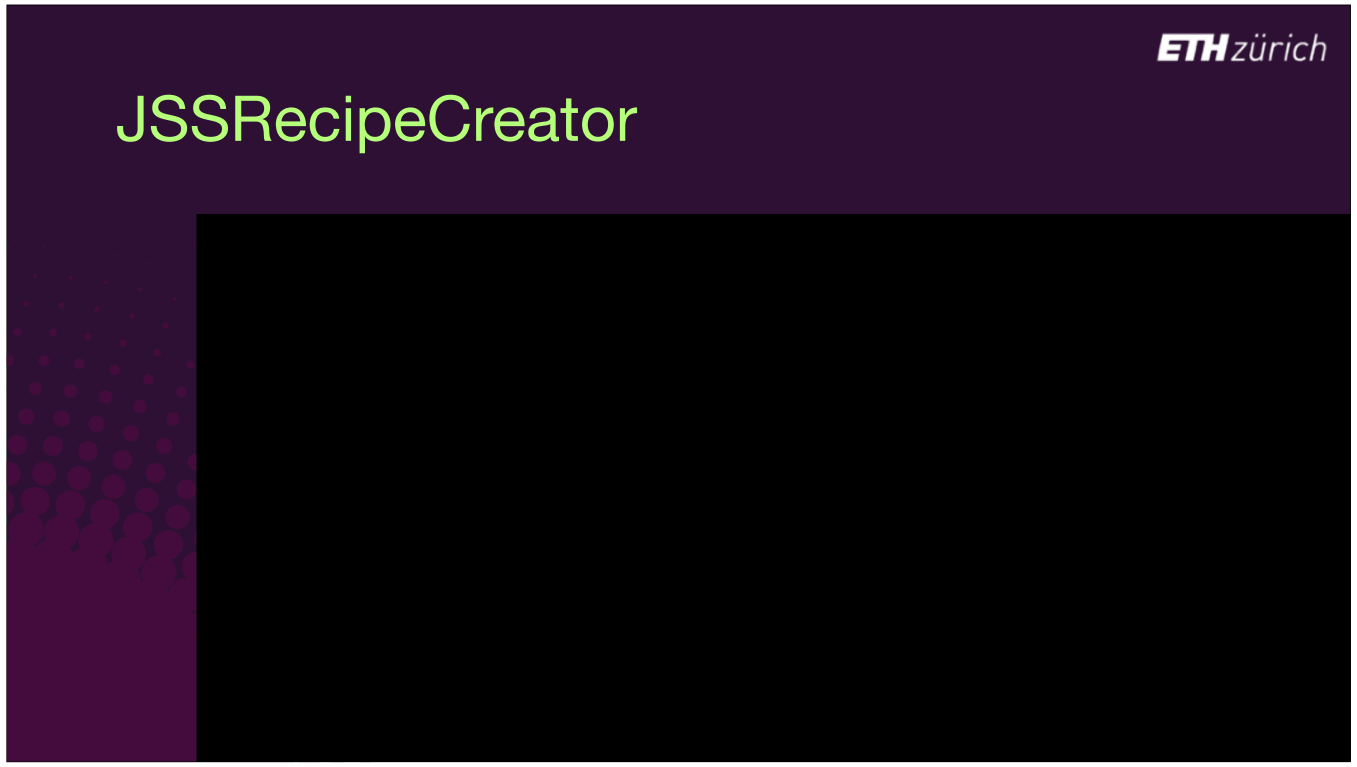
BrilliantApp.jss.recipe

© JAMF Software, LLC

If there are existing download and pkg recipes for your app,
[click] You only need to create your own JSS recipe.
Of course, if a standard JSS recipe already exists, you can just copy that into your own repo and edit to make the changes you need.
Just make sure you give it a different Identifier.

Bespoke JSS Recipes

BrilliantApp.jss.recipe



JSSRecipeCreator

- If there's no JSS recipe yet for an app, you can use JSSRecipeCreator to easily make one
- This is a tool created by Shea Craig who wrote JSSImporter.
- [click] Run from the command line with the auto option,
- [click] and specify the path to a parent pkg recipe.
- It will ask you what package category you want, and then generate the recipe for you.

JSSRecipeCreator

```
$ JSSRecipeCreator.py
```

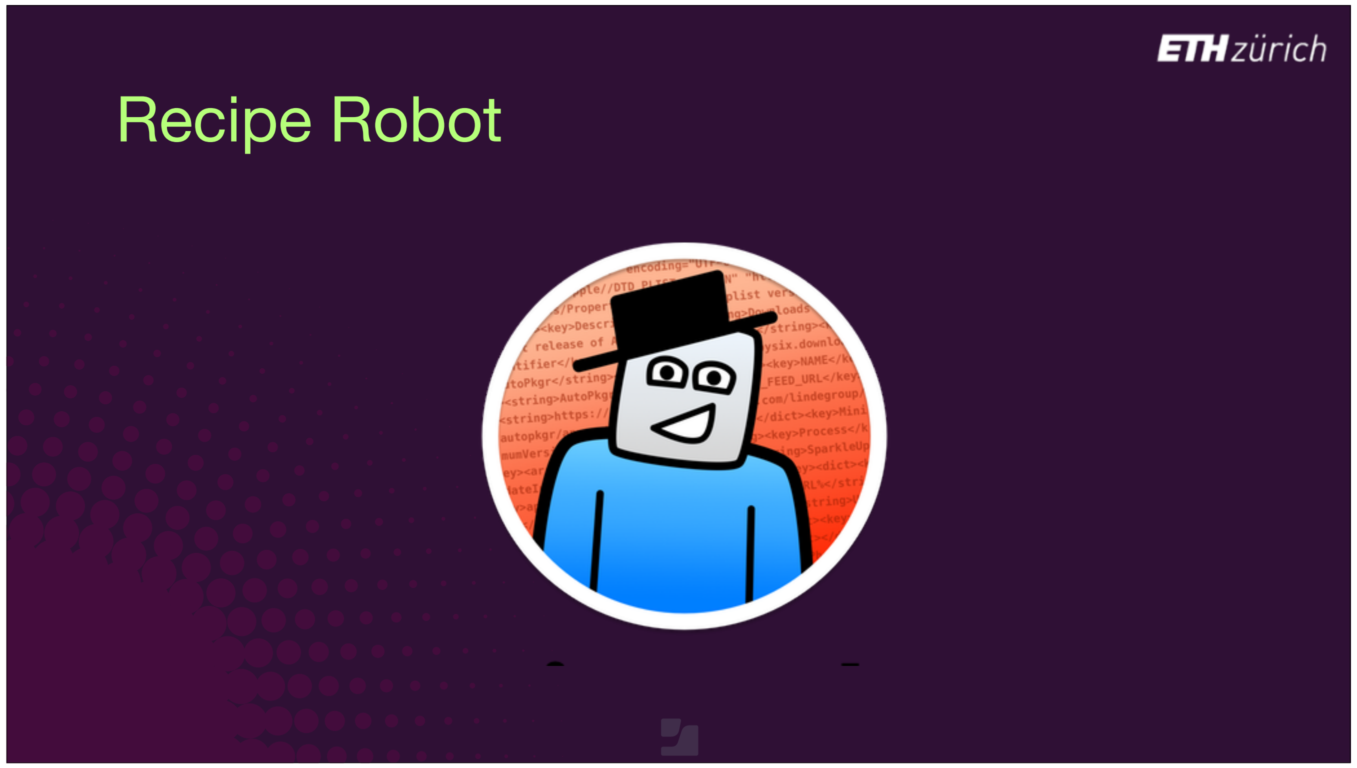
JSSRecipeCreator

```
$ JSSRecipeCreator.py --auto
```

ETH zürich

JSSRecipeCreator

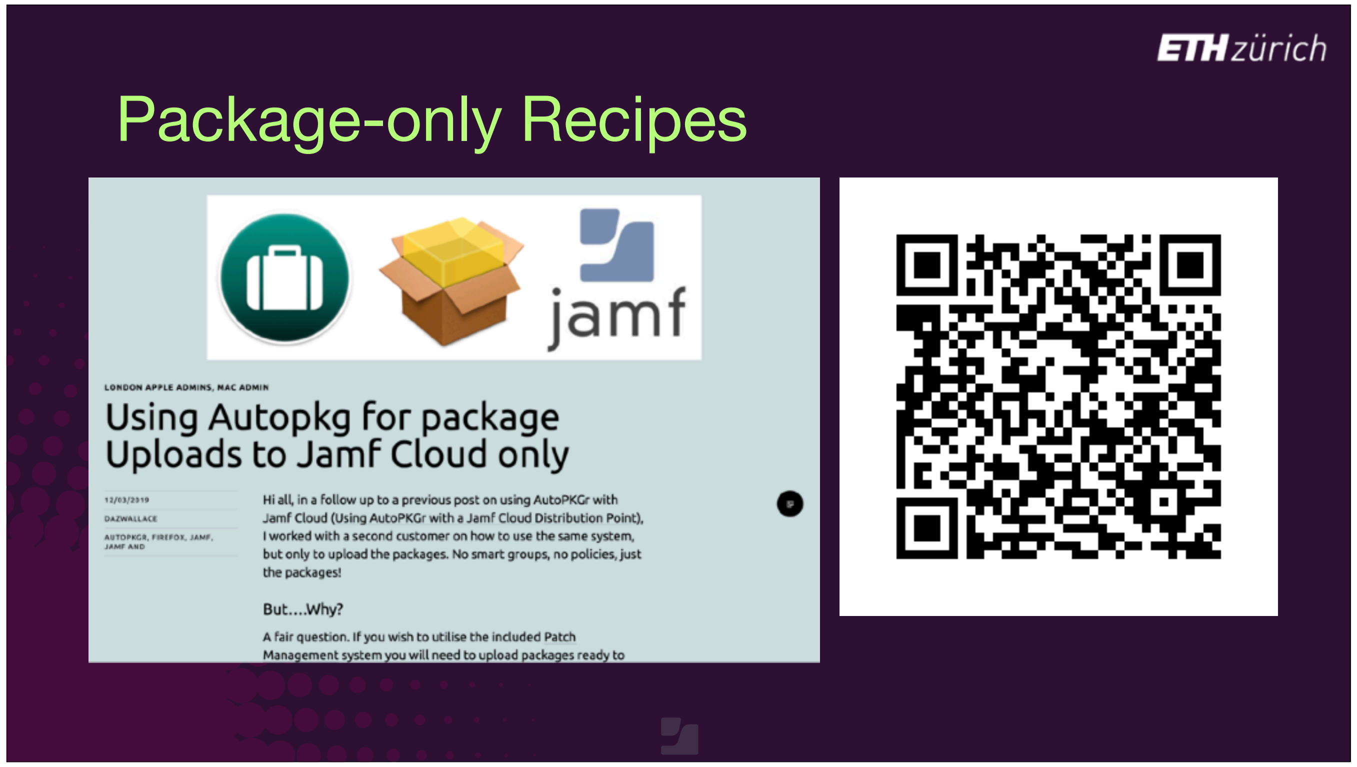
```
$ JSSRecipeCreator.py --auto
  ~/Library/AutoPkg/RecipeRepos/
  com.github.autopkg.recipes/VLC/VLC.pkg.recipe
```



ETHzürich

Recipe Robot

- [slide was not shown in the session]
- I should also give a shout out to Elliot Jordan's awesome app named Recipe Robot.
- This tool is primarily useful for building standard download, pkg, Munki, install and JSS recipes.
- Where just the JSS recipe is required, or you need a package-only recipe, I recommend JSSRecipeCreator instead.



ETH zürich

Package-only Recipes

LONDON APPLE ADMINS, MAC ADMIN

Using Autopkg for package Uploads to Jamf Cloud only

12/03/2019

DAZWALLACE

AUTOPKGR, FIREFOX, JAMF, JAMF AND

Hi all, in a follow up to a previous post on using AutoPKGr with Jamf Cloud (Using AutoPKGr with a Jamf Cloud Distribution Point), I worked with a second customer on how to use the same system, but only to upload the packages. No smart groups, no policies, just the packages!

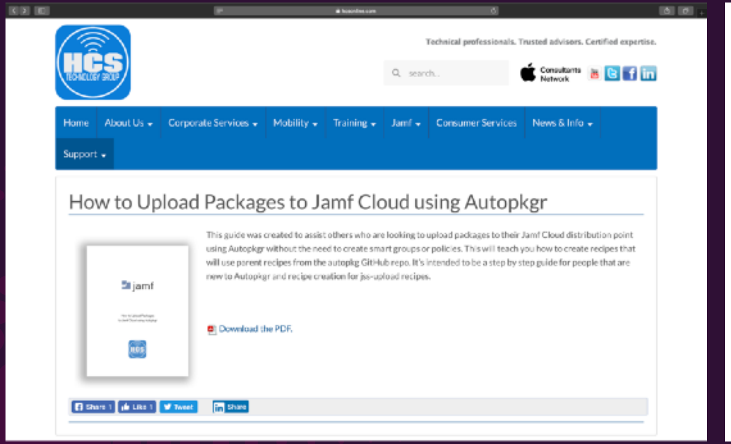
But….Why?

A fair question. If you wish to utilise the included Patch Management system you will need to upload packages ready to

- One of the most common questions I've been asked about is using AutoPkg to import packages without creating any policies. To do this, you currently need to create your own recipes.
- A use case for these is integrating AutoPkg with Patch Management - those of you in the AutoPkg workshop on Tuesday will have come across this.
- Daz Wallace of DataJAR has a blog post about creating these package-only recipes.
- He suggests that we give these recipes the .jss-upload suffix rather than just .jss, to avoid confusion with the standard recipes.

ETH zürich

Package-only Recipes



How to Upload Packages to Jamf Cloud using Autopkgr

This guide was created to assist others who are looking to upload packages to their Jamf Cloud distribution point using Autopkgr without the need to create smart groups or policies. This will teach you how to create recipes that will use parent recipes from the autopkg GitHub repo. It's intended to be a step by step guide for people that are new to Autopkgr and recipe creation for jss-upload recipes.

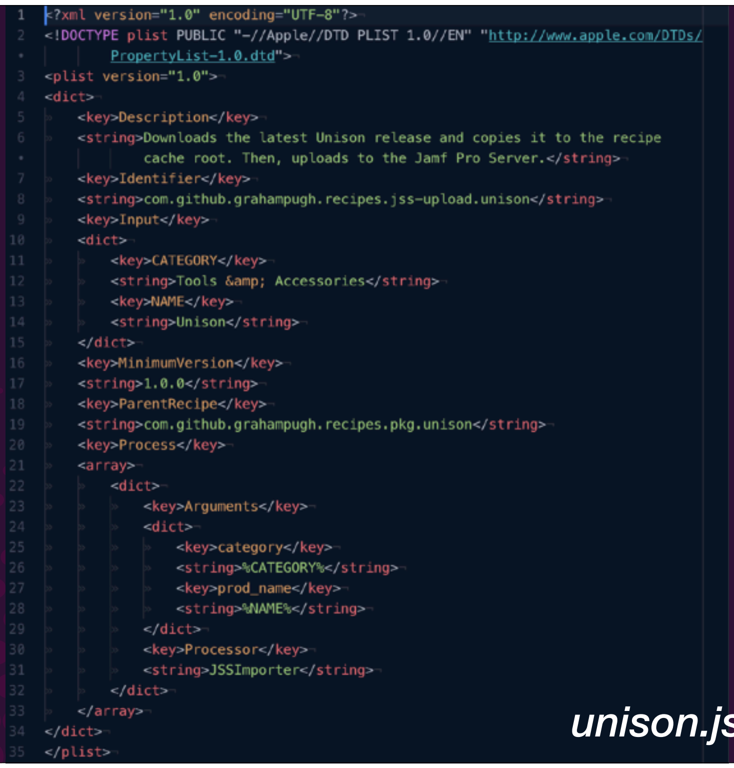
Download the PDF.

- [slide was not shown in the session]
- Keith Mitnick of HCS Technology Group also wrote about this concept, and explains how to use these recipes in AutoPkgr.

ETH zürich

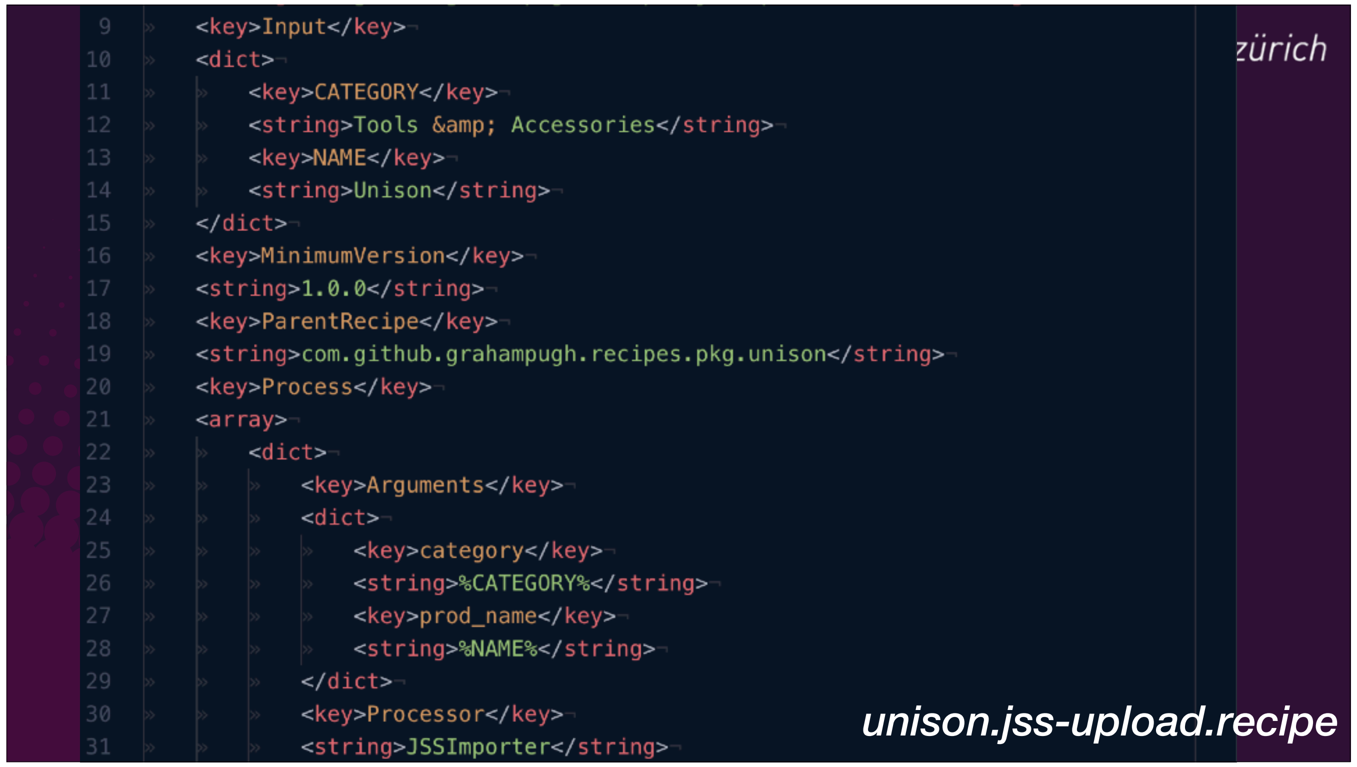
unison.jss-upload.recipe

- Let's look at an example package-only recipe for an app called Unison.
- [click] Since we don't want a policy or group, the recipe only needs a name, and a package category.
- [click] The Input array is therefore also very simple - just those two values.
- Pretty much all package-only recipes look the the same as this.



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
        PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>Description</key>
    <string>Downloads the latest Unison release and copies it to the recipe
            cache root. Then, uploads to the Jamf Pro Server.</string>
    <key>Identifier</key>
    <string>com.github.grahampugh.recipes.jss-upload.unison</string>
    <key>Input</key>
    <dict>
        <key>CATEGORY</key>
        <string>Tools &amp; Accessories</string>
        <key>NAME</key>
        <string>Unison</string>
    </dict>
    <key>MinimumVersion</key>
    <string>1.0.0</string>
    <key>ParentRecipe</key>
    <string>com.github.grahampugh.recipes.pkg.unison</string>
    <key>Process</key>
    <array>
        <dict>
            <key>Arguments</key>
            <dict>
                <key>category</key>
                <string>%CATEGORY%</string>
                <key>prod_name</key>
                <string>%NAME%</string>
            </dict>
            <key>Processor</key>
            <string>JSSImporter</string>
        </dict>
    </array>
</dict>
</plist>
```

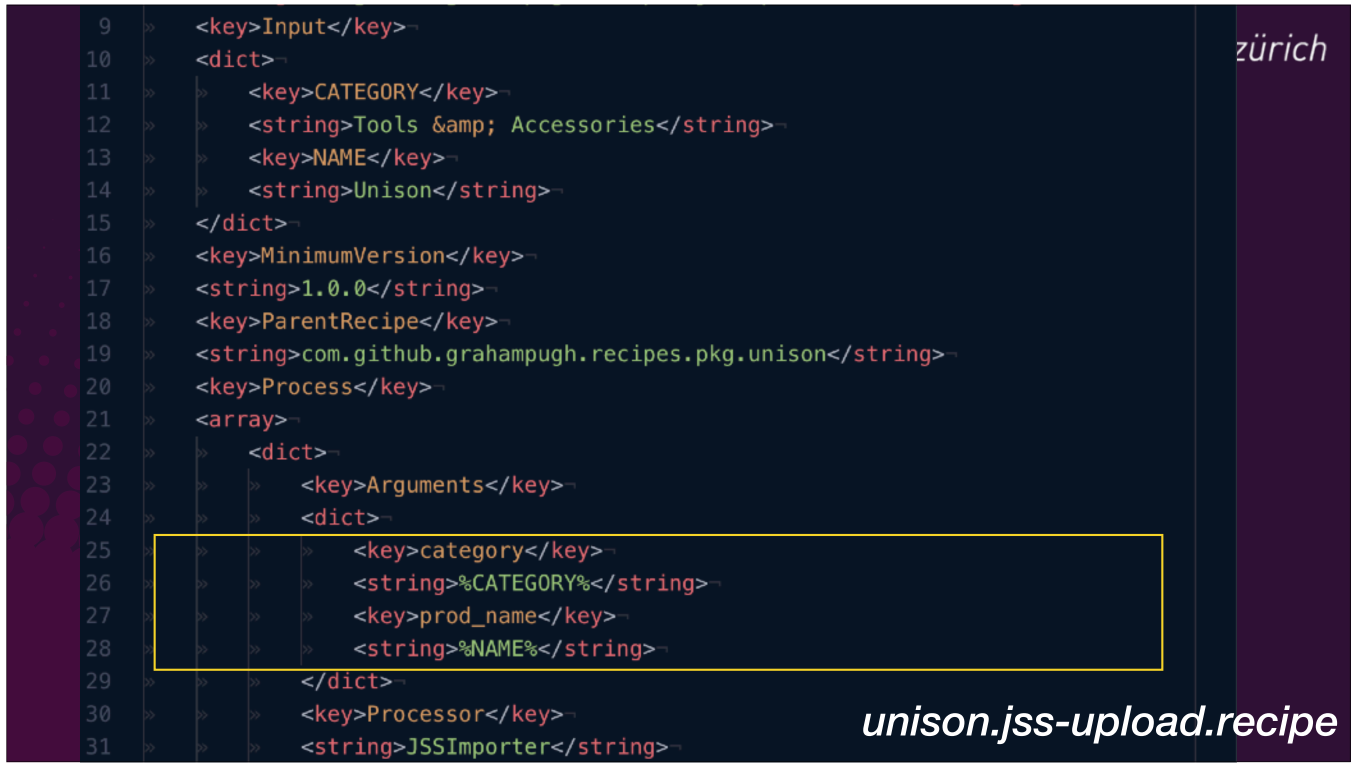
unison.jss-upload.recipe



```
	<key>Input</key>
	<dict>
		<key>CATEGORY</key>
		<string>Tools &amp; Accessories</string>
		<key>NAME</key>
		<string>Unison</string>
	</dict>
	<key>MinimumVersion</key>
	<string>1.0.0</string>
	<key>ParentRecipe</key>
	<string>com.github.grahampugh.recipes.pkg.unison</string>
	<key>Process</key>
	<array>
		<dict>
			<key>Arguments</key>
			<dict>
				<key>category</key>
				<string>%CATEGORY%</string>
				<key>prod_name</key>
				<string>%NAME%</string>
			</dict>
			<key>Processor</key>
			<string>JSSImporter</string>
```

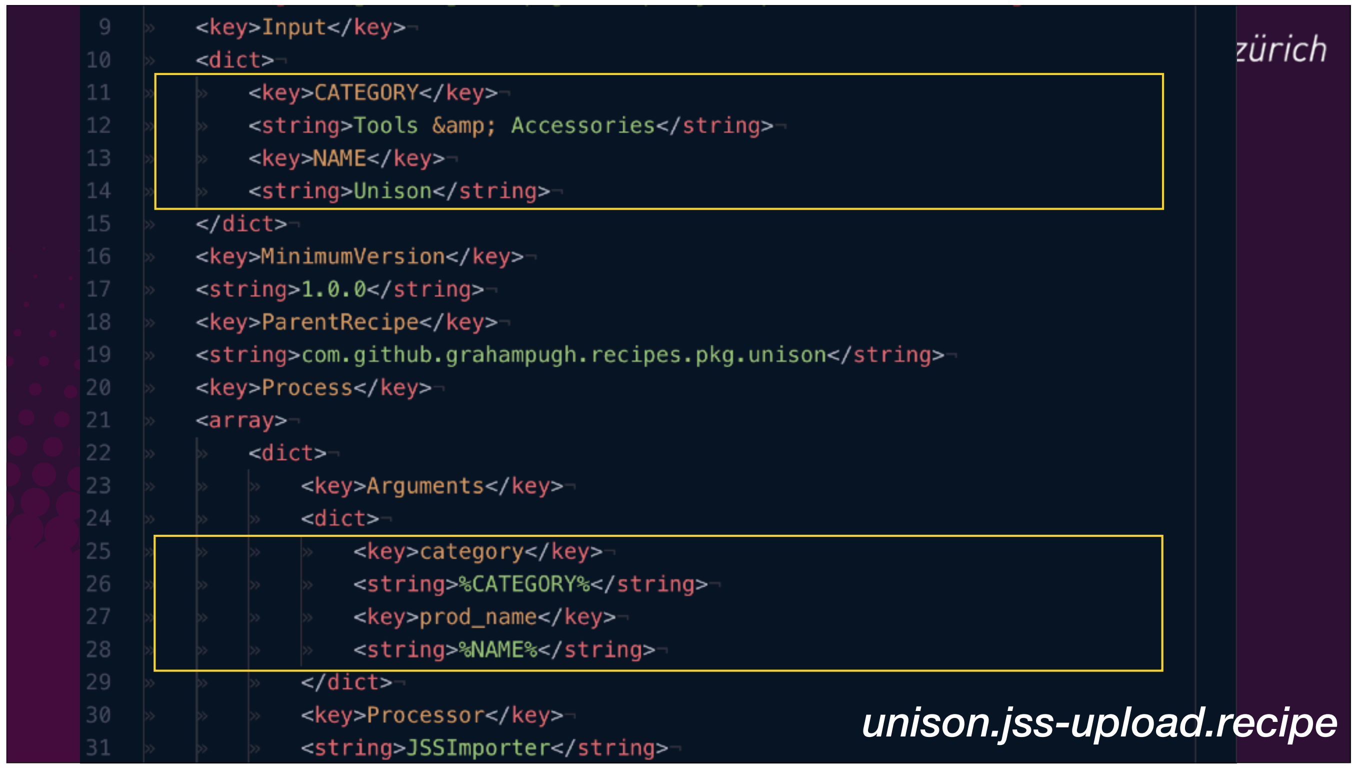
zürich

unison.jss-upload.recipe



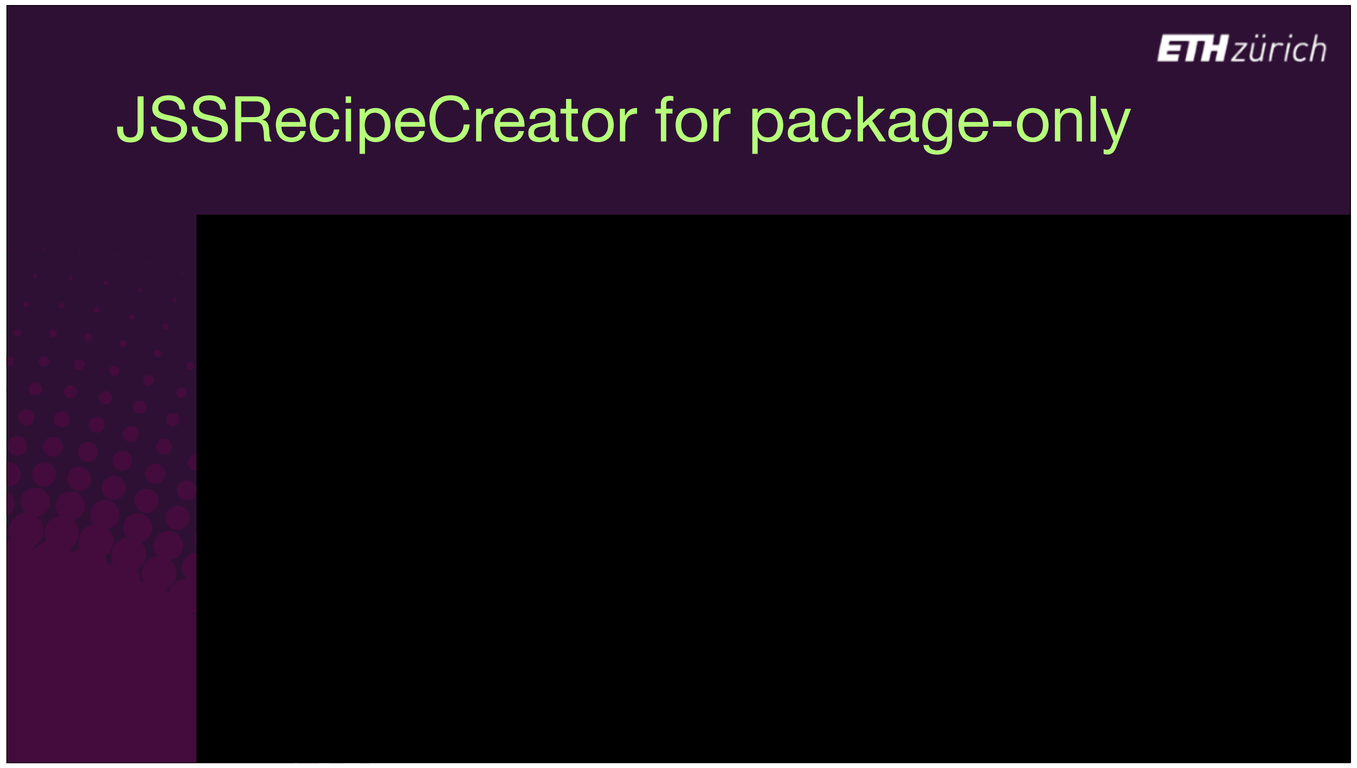
```
    <key>Input</key>
    <dict>
        <key>CATEGORY</key>
        <string>Tools &amp; Accessories</string>
        <key>NAME</key>
        <string>Unison</string>
    </dict>
    <key>MinimumVersion</key>
    <string>1.0.0</string>
    <key>ParentRecipe</key>
    <string>com.github.grahampugh.recipes.pkg.unison</string>
    <key>Process</key>
    <array>
        <dict>
            <key>Arguments</key>
            <dict>
                <key>category</key>
                <string>%CATEGORY%</string>
                <key>prod_name</key>
                <string>%NAME%</string>
            </dict>
            <key>Processor</key>
            <string>JSSImporter</string>
```

unison.jss-upload.recipe



```
	<key>Input</key>
	<dict>
		<key>CATEGORY</key>
		<string>Tools &amp; Accessories</string>
		<key>NAME</key>
		<string>Unison</string>
	</dict>
	<key>MinimumVersion</key>
	<string>1.0.0</string>
	<key>ParentRecipe</key>
	<string>com.github.grahampugh.recipes.pkg.unison</string>
	<key>Process</key>
	<array>
		<dict>
			<key>Arguments</key>
			<dict>
				<key>category</key>
				<string>%CATEGORY%</string>
				<key>prod_name</key>
				<string>%NAME%</string>
			</dict>
			<key>Processor</key>
			<string>JSSImporter</string>
```

unison.jss-upload.recipe



JSSRecipeCreator for package-only

- Since there has been recent interest in package-only recipes, I've also added the ability to automatically generate these types of recipes with JSSRecipeCreator.
- [click] Run from the command line with the package-only option,
- [click] and specify the path to a parent pkg recipe.
- [click] It will just ask you for the category, which you can choose from a list of existing categories on your Jamf Pro Server.

JSSRecipeCreator for package-only

```
$ JSSRecipeCreator.py
```

ETH zürich

JSSRecipeCreator for package-only

```
$ JSSRecipeCreator.py --package_only
```

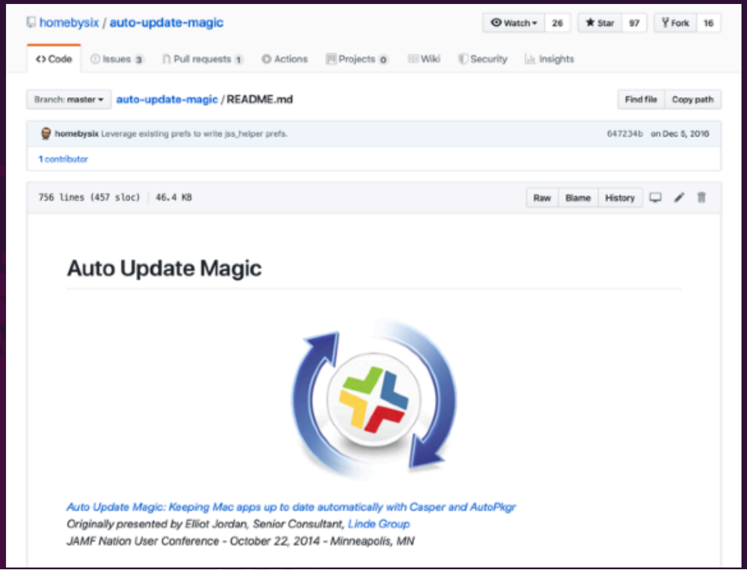
ETH zürich

JSSRecipeCreator for package-only

```
$ JSSRecipeCreator.py --package_only
  ~/Library/AutoPkg/RecipeRepos/
  com.github.autopkg.recipes/VLC/VLC.pkg.recipe
```

ETH zürich

Auto Update Magic - @homebysix


homebysix / auto-update-magic

Watch 26 | Star 97 | Fork 16

Branch: master auto-update-magic / README.md

homebysix Leverage existing prefs to write jss_helper prefs. 647234b on Dec 5, 2016

1 contributor

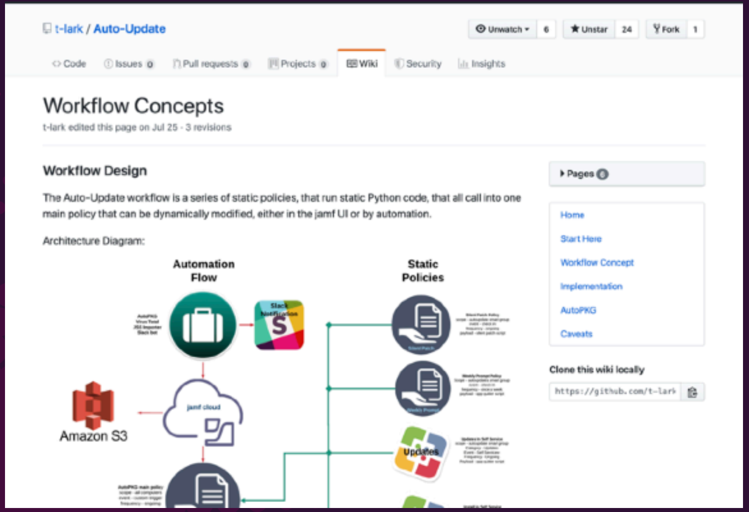
756 lines (457 sloc) | 46.4 KB

Auto Update Magic

Auto Update Magic: Keeping Mac apps up to date automatically with Casper and AutoPkgr
Originally presented by Elliot Jordan, Senior Consultant, Linde Group
JAMF Nation User Conference - October 22, 2014 - Minneapolis, MN

- What about using recipes to create your prod policies?
- You might have heard of Elliot Jordan's Auto Update Magic, which was presented here at JNUC back in 2014.
- This is a complete testing to prod workflow which uses standard JSS recipes for testing, and a different set of recipes for prod.
- Production recipes are run manually when you are ready to promote.
- He provides some examples, but for most applications you need to create your own production recipes and templates.
- I don't have enough time to go into it here but he explains how to do it in the wiki.

Auto Update - @tlark


t-lark / Auto-Update

Unwatch 6 | Unstar 24 | Fork 1

Code | Issues 0 | Pull requests 0 | Projects 0 | Wiki | Security | Insights

Workflow Concepts

t-lark edited this page on Jul 25 · 3 revisions

Workflow Design

The Auto-Update workflow is a series of static policies, that run static Python code, that all call into one main policy that can be dynamically modified, either in the jamf UI or by automation.

Architecture Diagram:

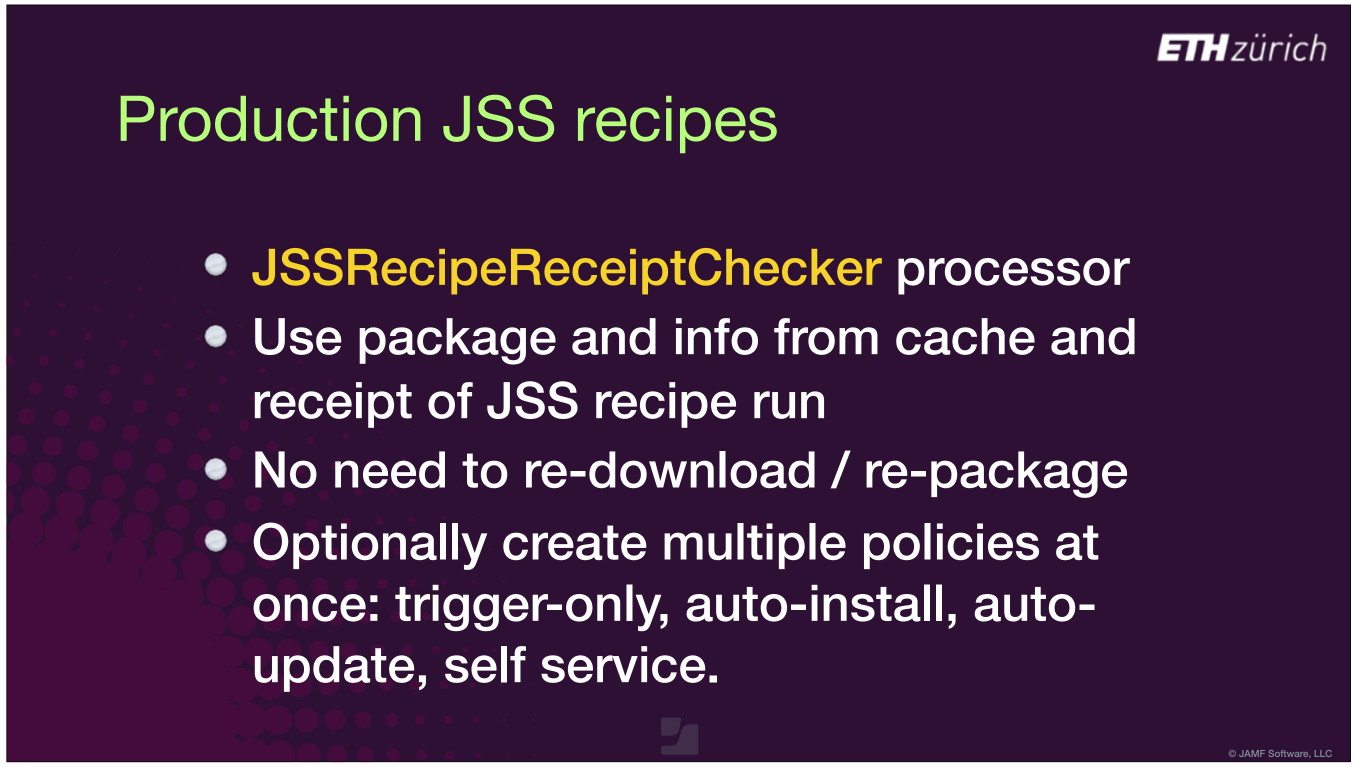
Pages 6

Home
Start Here
Workflow Concept
Implementation
AutoPKG
Caveats

Clone this wiki locally

https://github.com/t-lark

- [slide was not shown in the session]
- Tom Larkin wrote a good workflow for creating your production policies.
- This describes a similar workflow to what I just described, with some additional features to scope smartly, and also some extra magic to ensure that an app is not running before updating.
- There is manual work to generate the policies in the first place, but for most organisations with a single Jamf Pro instance, and not too many apps, the amount of manual work to set up and maintain the workflow is perfectly manageable.



Production JSS recipes

- JSSRecipeReceiptChecker processor
- Use package and info from cache and receipt of JSS recipe run
- No need to re-download / re-package
- Optionally create multiple policies at once: trigger-only, auto-install, auto-update, self service.

© JAMF Software, LLC

- [slide was not shown in the session]
- I use a similar concept, but I wanted to use the cache of the standard JSS recipe run rather than download it again.
- [click] So I wrote a processor to read the receipt from the last run of the JSS recipe.
- [click] All AutoPkg recipe runs create a receipt file that logs everything that happened during the run.
- The processor grabs the package name, category and self service description from the receipt, and creates or updates the existing prod policy direct from AutoPkg – no other tools required.
- [click] There is no parent recipe, so no need to re-download and repackage the app. You already have the package you want.
- [click] If you want, these recipes can include multiple processors to create different policy types for different clients.
- This is particularly useful when creating more advanced policies and scopes for more difficult applications.

Script-only recipes

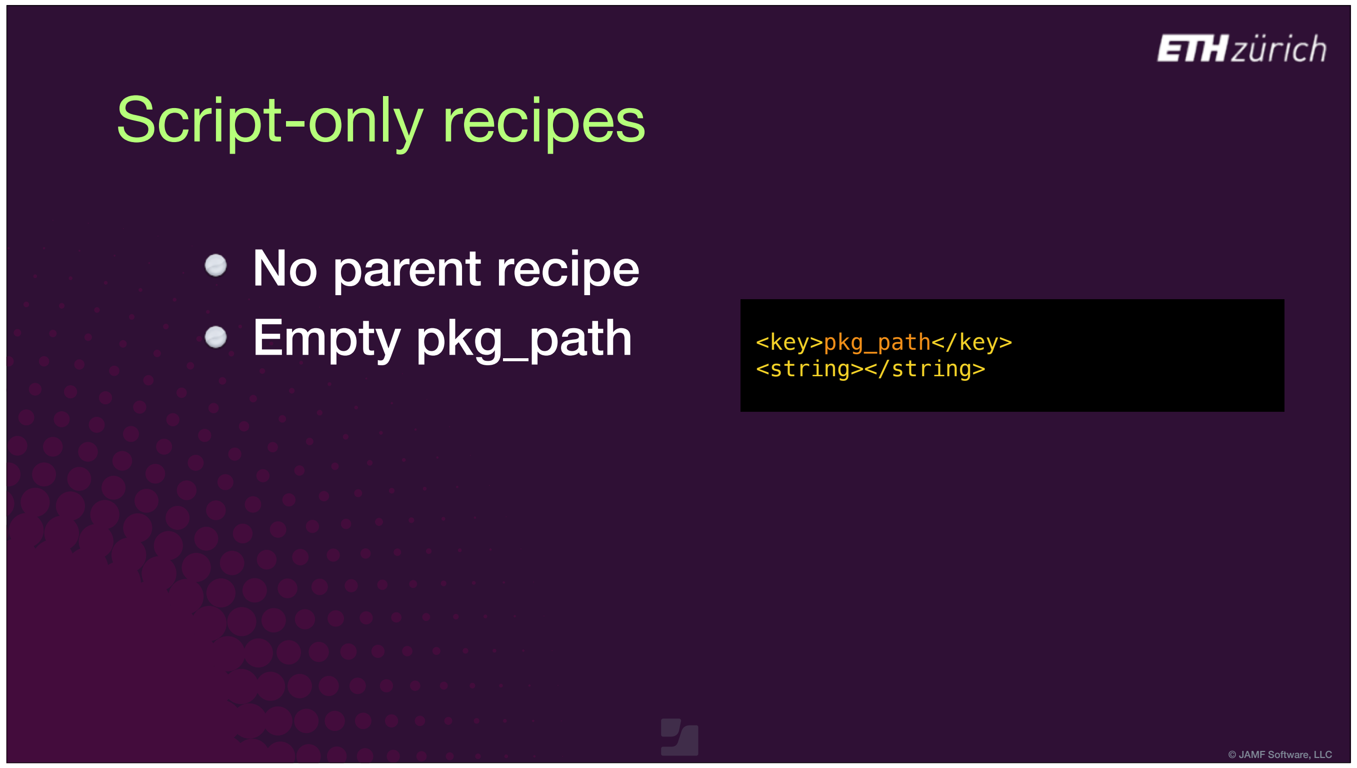
ETH zürich

© JAMF Software, LLC

- We also have a bunch of policies in our Jamf Pro server that don't have packages.
- We have script-based policies for all sorts of things, like installing printers, uninstalling applications, ignoring updates and so on.
- JSSImporter can be used to create any type of policy!
- [click] Script-only policies are just like normal JSS recipes, but they don't need a parent recipe, since there's no pkg.
- [click] they have an empty pkg_path key in the Processor instructions,
- [click] and require a scripts array, where the script is given a name and a template file.
- Depending on the purpose of the script, you might want to make a once-per-computer policy, or scope it based on an Extension Attribute, which can also be bundled in with the recipe.
- [click] This way, we can have everything that's in our Jamf Pro server in git, and every type of policy can be added and updated using AutoPkg.
- I have some examples of script only recipes in my recipe repo to get you started, and my intention is to make more available, particularly for maintaining uninstaller policies.

Script-only recipes

- No parent recipe



Script-only recipes

- No parent recipe
- Empty pkg_path

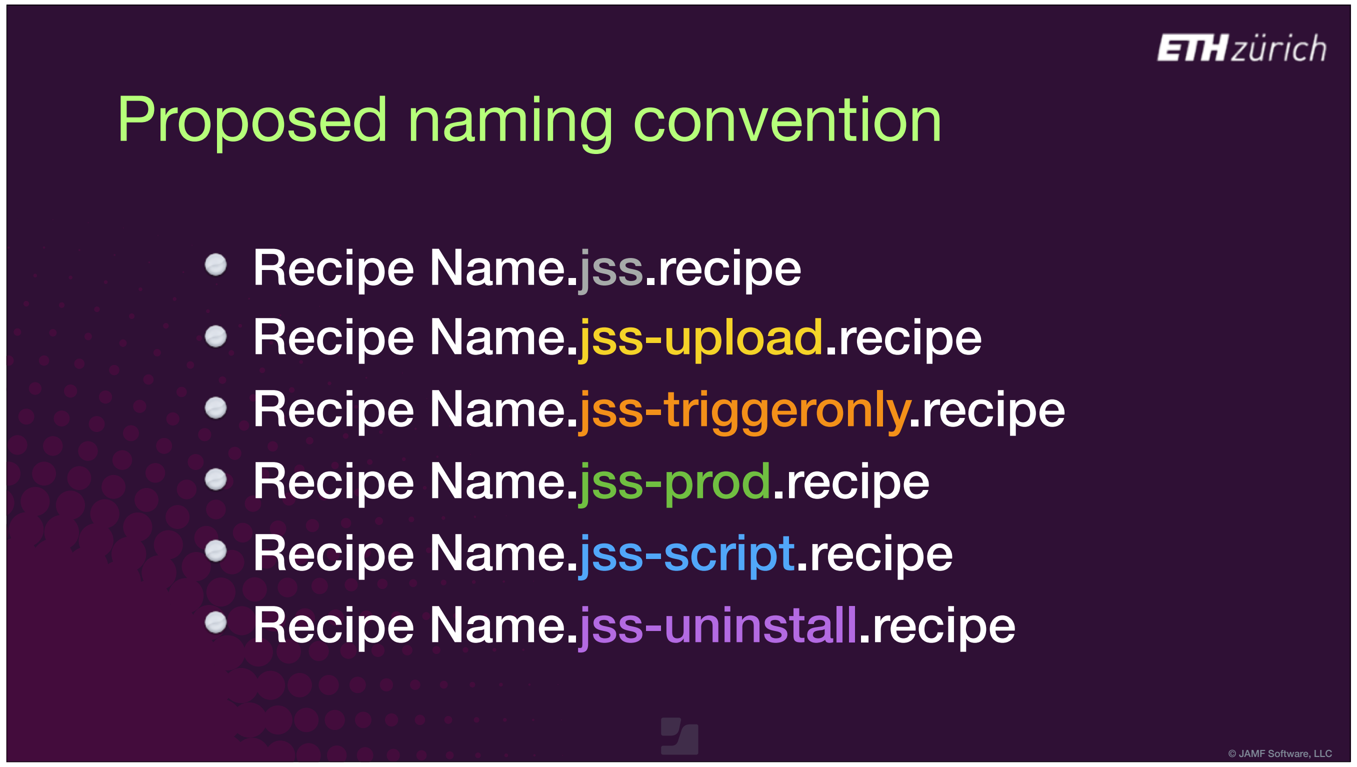
```
<key>pkg_path</key>
<string></string>
```

ETH zürich

Script-only recipes

- No parent recipe
- Empty pkg_path
- Array of scripts

```
<key>pkg_path</key>
<string></string>

<key>scripts</key>
<array>
 <dict>
  <key>name</key>
  <string>%SCRIPT_NAME%.sh</string>
  <key>template_path</key>
  <string>%SCRIPT_TEMPLATE%</string>
 </dict>
</array>
```

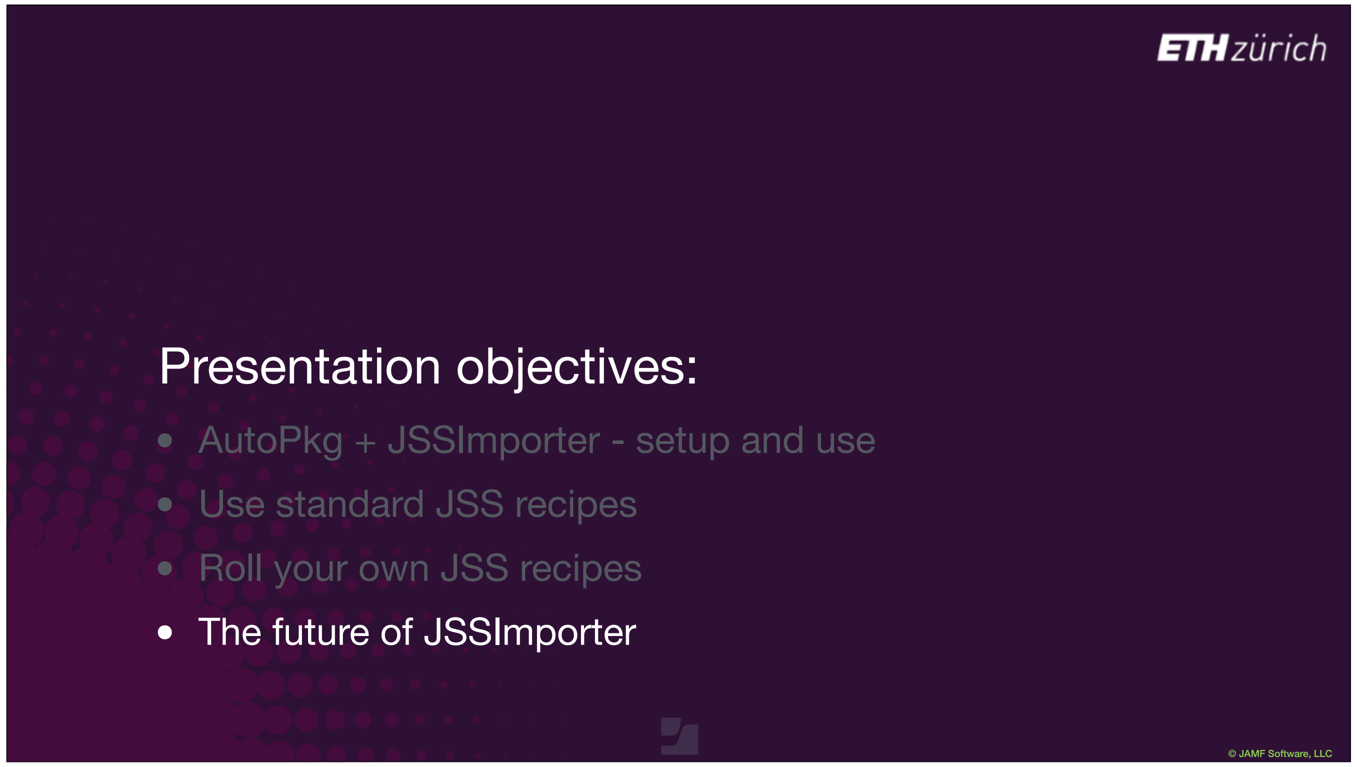


Proposed naming convention

- Recipe Name.jss.recipe
- Recipe Name.jss-upload.recipe
- Recipe Name.jss-triggeronly.recipe
- Recipe Name.jss-prod.recipe
- Recipe Name.jss-script.recipe
- Recipe Name.jss-uninstall.recipe

© JAMF Software, LLC

- [slide was not shown in the session]
- I've talked about a bunch of random ideas for using AutoPkg and JSSImporter.
- They're all serving different needs.
- None of these different types of recipes have really been shared until now, but if we do, we should have a naming convention.
- [click] Only the standard testing recipes should be called JSS recipes.
- [click] Daz Wallace suggested jss-upload for the package-only recipes.
- I think that was a good idea. Let's extend that to other types of recipe.
- I am using some of these in my organisation, and you'll already see some examples in my autopkg recipe repo.



Presentation objectives:

- AutoPkg + JSSImporter - setup and use
- Use standard JSS recipes
- Roll your own JSS recipes
- The future of JSSImporter

Finally let's take a look at the future of JSSImporter.

Stop!

- [slide was not shown in the session]

A big change I made is to change the way that JSSImporter handles repeat AutoPkg runs.

- JSSImporter used to update the policy, script, extension attributes and groups every time AutoPkg ran, even if there was no change to the package.
- This is a problem if you have a workflow where you delete your Testing policies when your app moves to production.
- To prevent this, I introduced a new STOP_IF_NO_JSS_UPLOAD key to JSSImporter.
- It's a True or False key, set to True by default.
- Now, if the package is unchanged from the last time AutoPkg ran, the processor just stops, and no changes are made to your server.
- If you don't like this change, and want to overwrite your policies every AutoPkg run...
- [click] run AutoPkg with the key set to False,
- [click] or, you can add the key in your recipe override
- [click] or add it to you your AutoPkg preferences file to revert to the old behaviour for all recipes.

ETH zürich

Stop!

```
STOP_IF_NO_JSS_UPLOAD
```

ETH zürich

Stop!

```
STOP_IF_NO_JSS_UPLOAD
```

```
$ autopkg run "Brilliant App.jss"
  --key STOP_IF_NO_JSS_UPLOAD=False
```

ETH zürich

Stop!

```
STOP_IF_NO_JSS_UPLOAD
```

```
$ autopkg run "Brilliant App.jss"
  --key STOP_IF_NO_JSS_UPLOAD=False
```

```
  <key>STOP_IF_NO_JSS_UPLOAD</key>
  <false/>
```

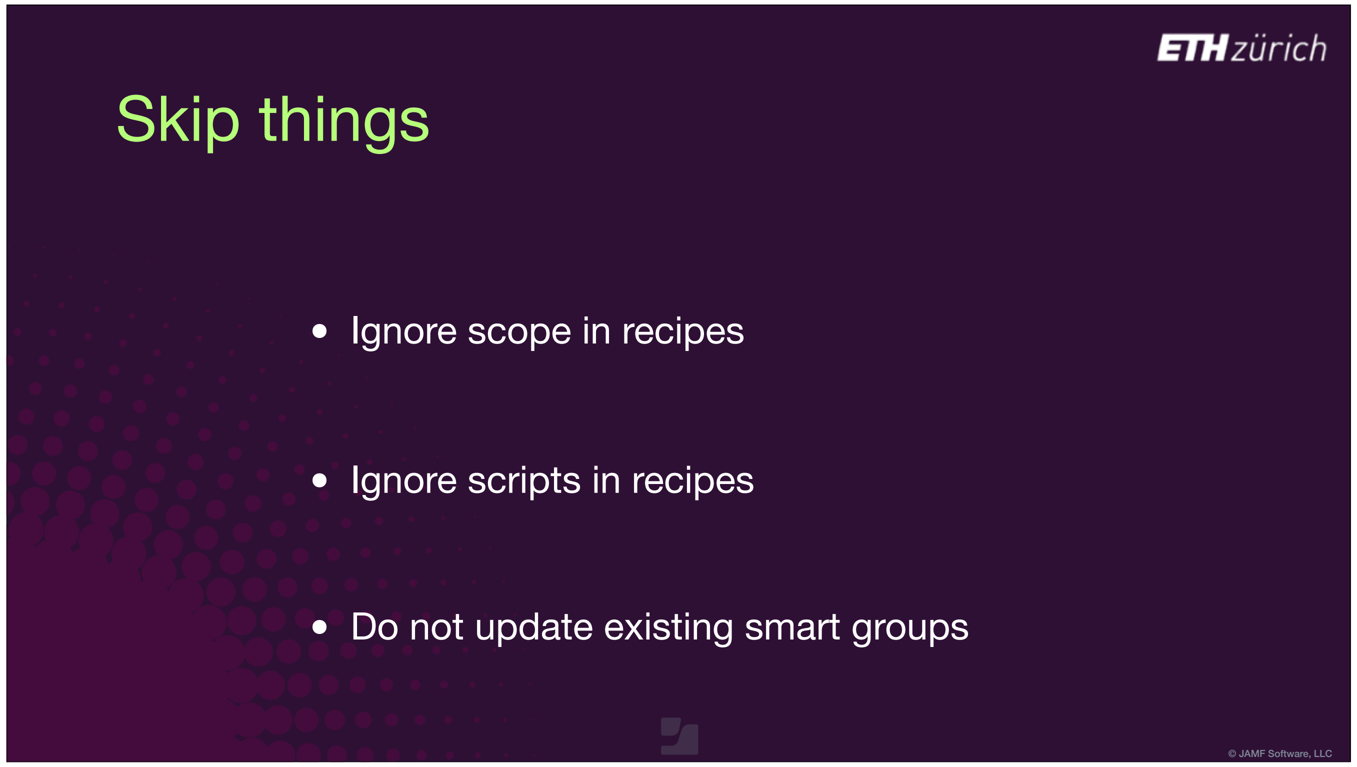
Stop!

```
STOP_IF_NO_JSS_UPLOAD
```

```
$ autopkg run "Brilliant App.jss"
  --key STOP_IF_NO_JSS_UPLOAD=False
```

```
  <key>STOP_IF_NO_JSS_UPLOAD</key>
  <false/>
```

```
$ defaults write com.github.autopkg
  STOP_IF_NO_JSS_UPLOAD -bool false
```



Skip things

- Ignore scope in recipes
- Ignore scripts in recipes
- Do not update existing smart groups

- [slide was not shown in the session]

Here are some other new changes I don't have time to describe, which give you more flexibility on deploying production policies. I'll include more details in the link at the end of the talk.

ETH zürich

Skip things

```
skip_scope = True
```

- Ignore scope in recipes
- Ignore scripts in recipes
- Do not update existing smart groups

ETH zürich

Skip things

```
skip_scope = True
```

- Ignore scope in recipes

```
skip_scripts = True
```

- Ignore scripts in recipes

- Do not update existing smart groups

ETH zürich

Skip things

```
skip_scope = True
```

- Ignore scope in recipes

```
skip_scripts = True
```

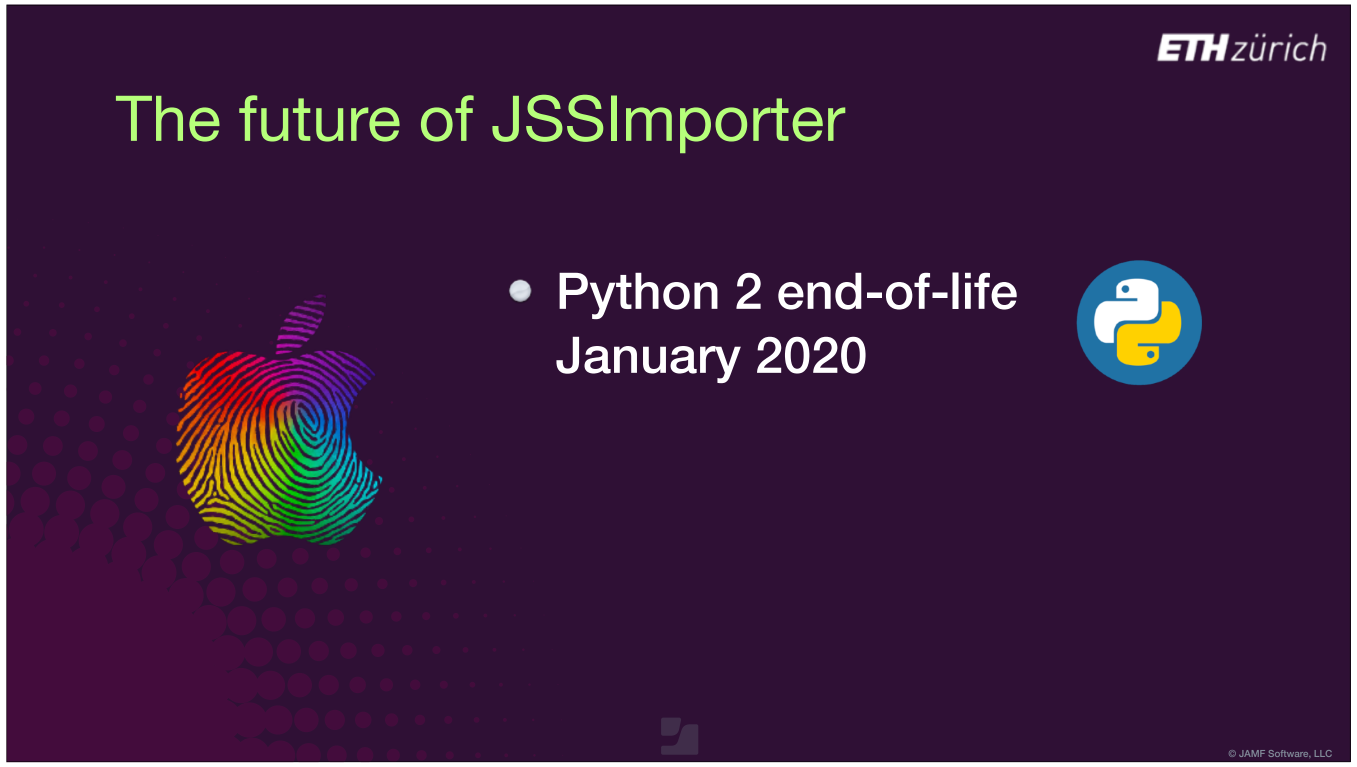
- Ignore scripts in recipes

```
do_update = False
```

- Do not update existing smart groups

The future of JSSImporter

- JSSImporter's future depends on three big factors.
- The first one is Apple. Apple have bundled Python 2 in with macOS since the early days of OS 10.
- JSSImporter is a Python processor, dependent on an underlying Python framework called 'python-jss'.
- [click] But the python developers have declared that January 2020 is the end-of-life date for Python 2.
- [click] Apple have responded by stating they won't bundle any version of Python in with future versions of macOS, so AutoPkg users will have to install Python themselves in future.
- [click] Nick McSpadden of Facebook and others are currently refactoring AutoPkg to work on python 3, and we have to do the same for python-jss and JSSImporter.
- Mosen has done some of the work, but there's busy times ahead for the next few months...



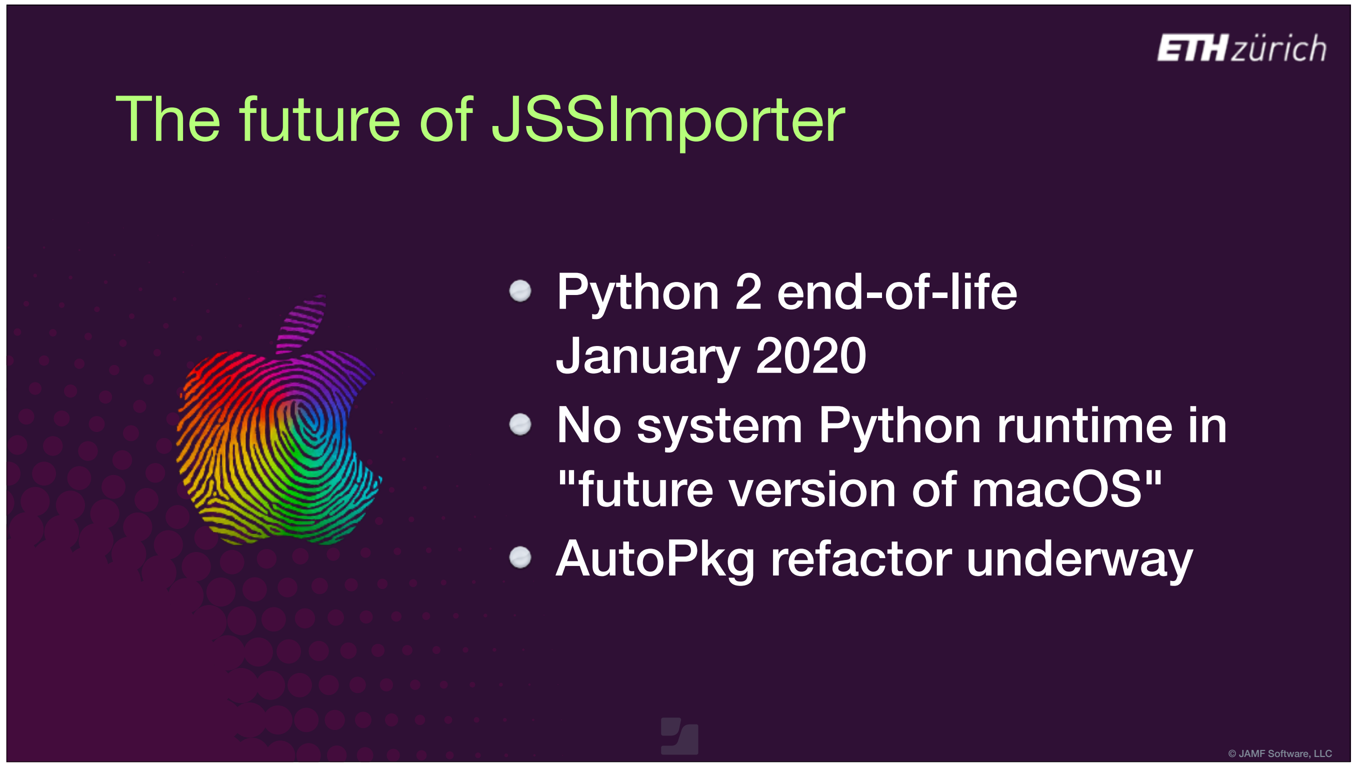
The future of JSSImporter

- Python 2 end-of-life January 2020

ETH zürich

The future of JSSImporter

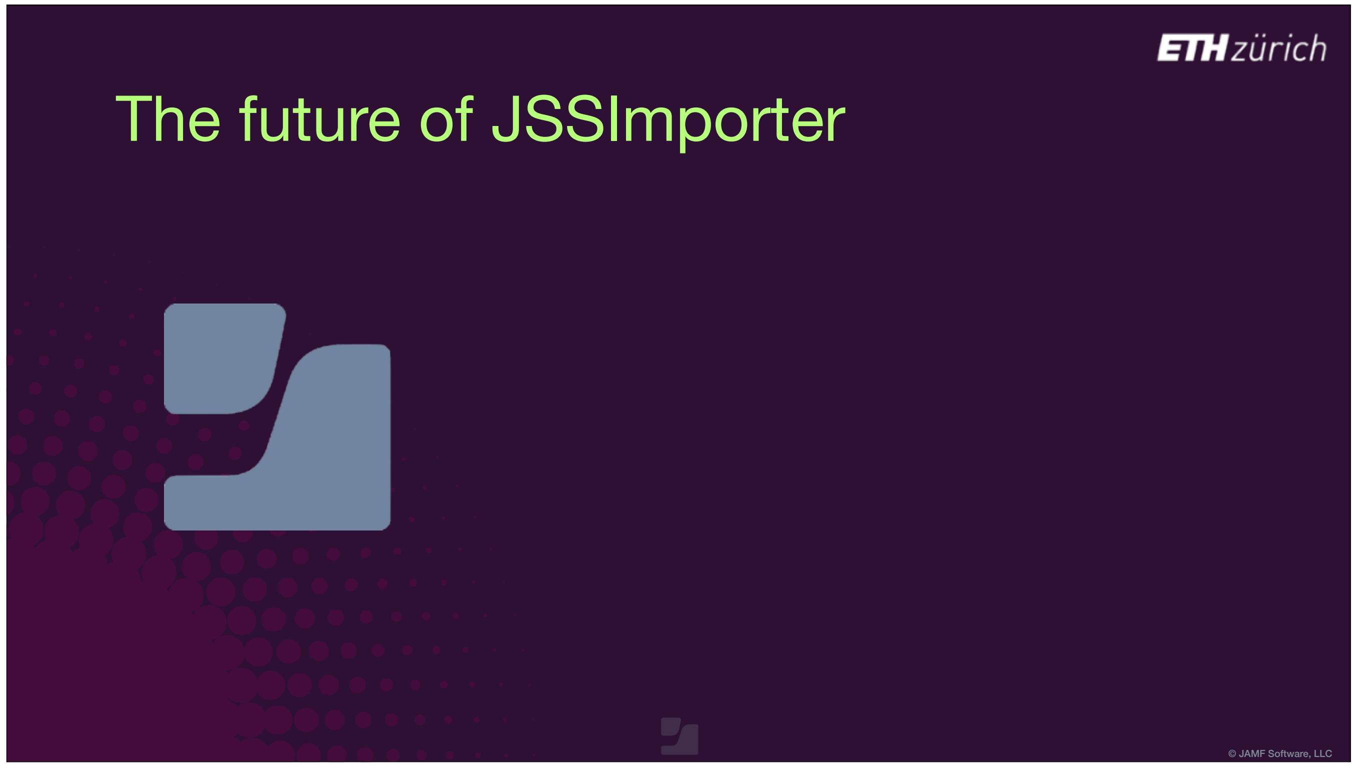
- Python 2 end-of-life January 2020
- No system Python runtime in "future version of macOS"



ETH zürich

The future of JSSImporter

- Python 2 end-of-life
  January 2020
- No system Python runtime in "future version of macOS"
- AutoPkg refactor underway

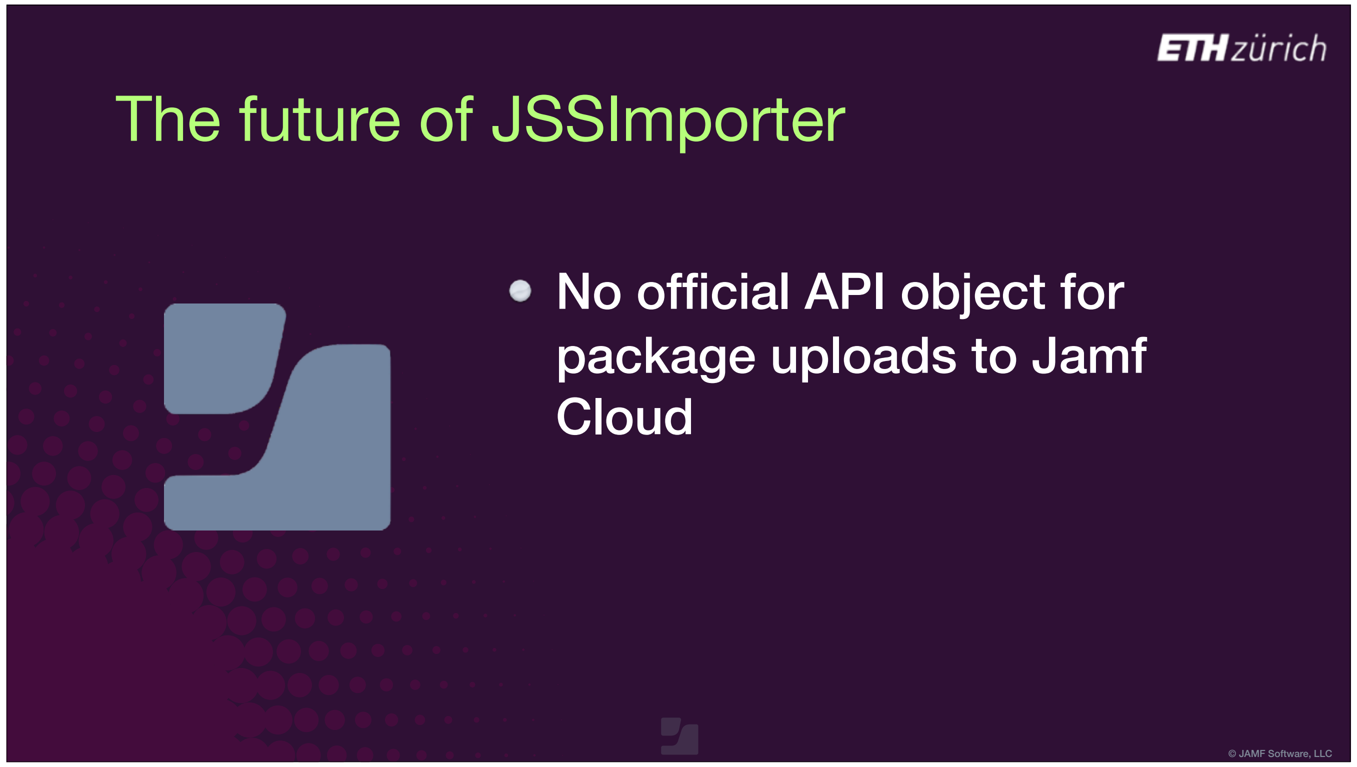


The future of JSSImporter

- The second thing we depend on is of course Jamf.
- Back in 2014 there was no Jamf Cloud. Packages were uploaded to a repo using SMB or AFP.
- [click] But when the Jamf Distribution Server was created, no official API object was documented for uploading packages.
- Shea Craig was able to reverse engineer the Casper Admin app to reveal an undocumented API request object for uploading packages, and build this into python-jss so that JSSImporter would work with the JDS, but it's always been a hack.
- [click] Unfortunately, Jamf Cloud also has no official documented API object for uploading packages. We are lucky that the JDS method is also more-or-less working with Jamf Cloud, but it's pretty fragile.

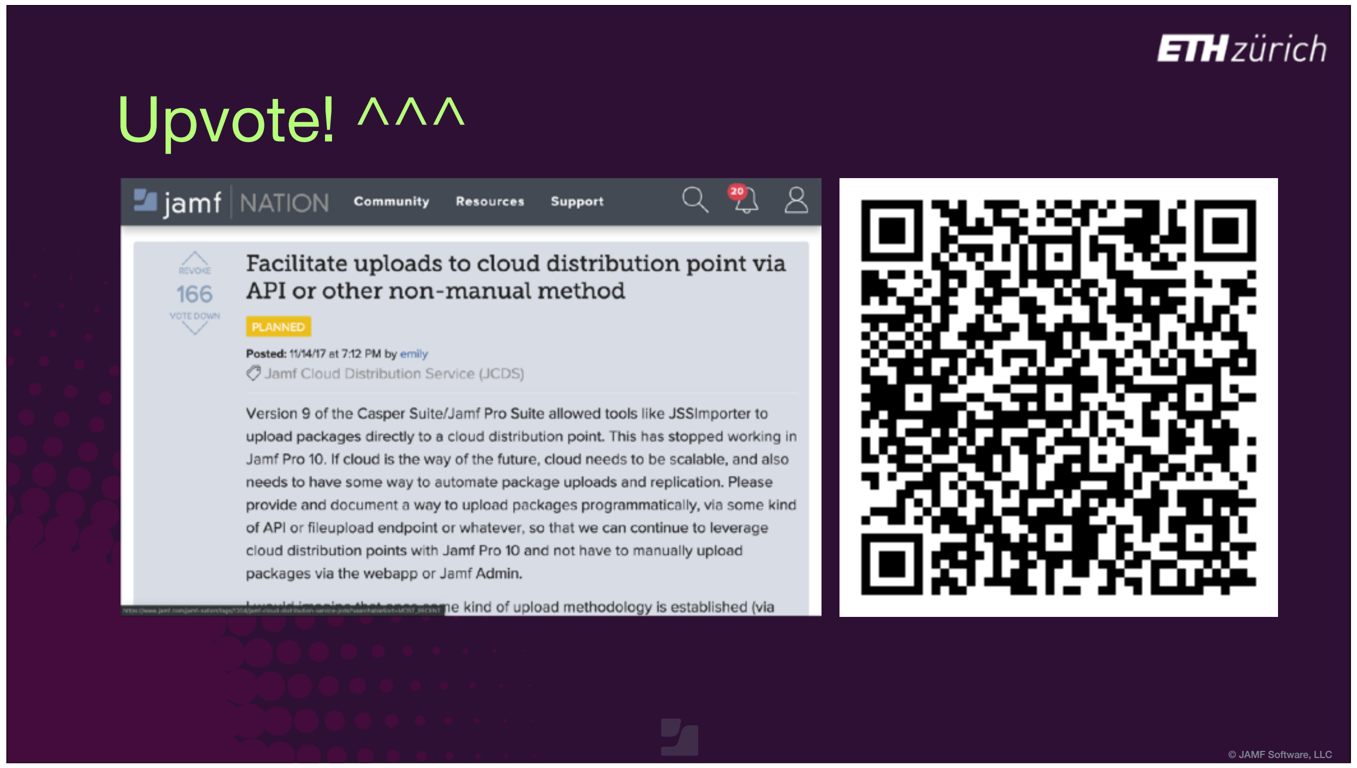
ETH zürich

The future of JSSImporter



The future of JSSImporter

- No official API object for package uploads to Jamf Cloud



Upvote! ^^^

ETH zürich

jamf | NATION Community Resources Support

REVOTE 166 VOTE DOWN

Facilitate uploads to cloud distribution point via API or other non-manual method

PLANNED

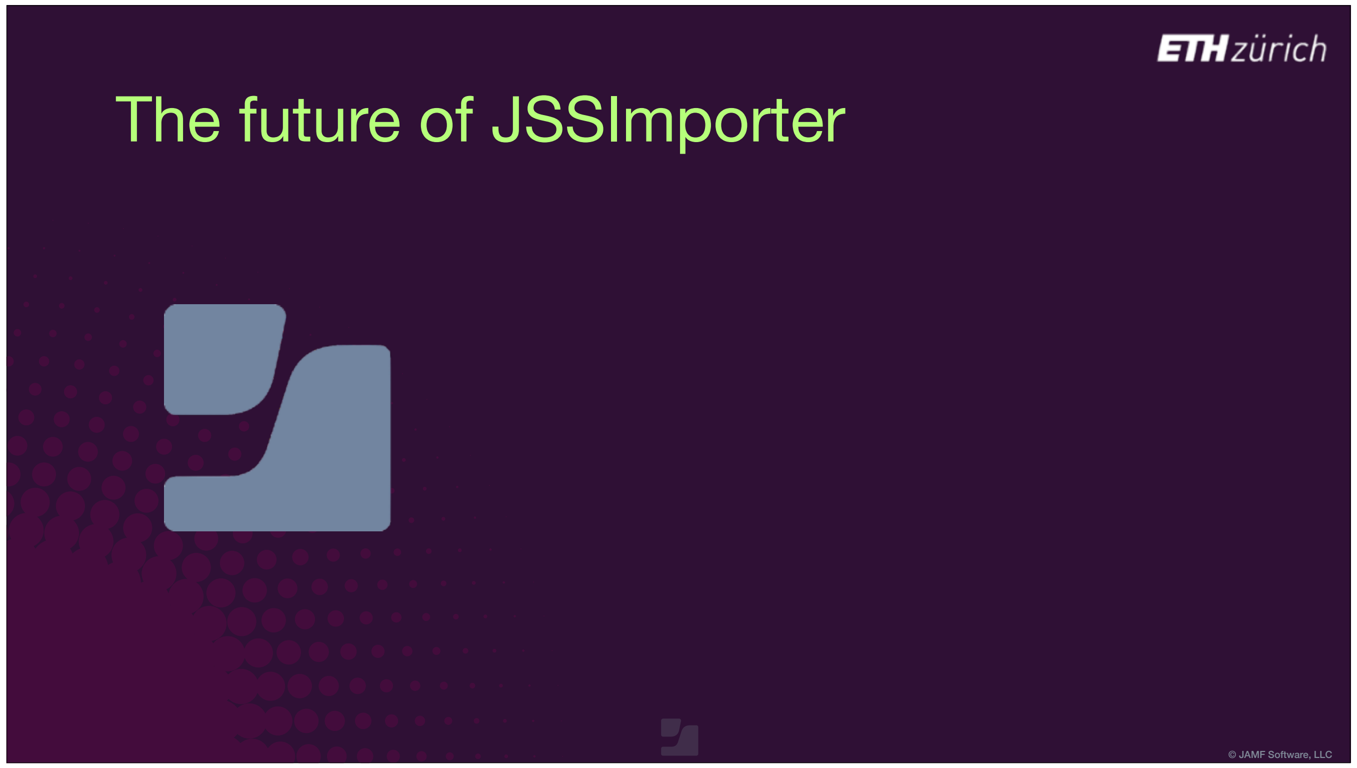
Posted: 11/14/17 at 7:12 PM by emily

Jamf Cloud Distribution Service (JCDS)

Version 9 of the Casper Suite/Jamf Pro Suite allowed tools like JSSImporter to upload packages directly to a cloud distribution point. This has stopped working in Jamf Pro 10. If cloud is the way of the future, cloud needs to be scalable, and also needs to have some way to automate package uploads and replication. Please provide and document a way to upload packages programmatically, via some kind of API or fileupload endpoint or whatever, so that we can continue to leverage cloud distribution points with Jamf Pro 10 and not have to manually upload packages via the webapp or Jamf Admin.

I would imagine that once some kind of upload methodology is established (via

- I've had good discussions with Jamf about this problem this week, but please do upvote this Feature Request to give the issue more impact!


ETH zürich

The future of JSSImporter

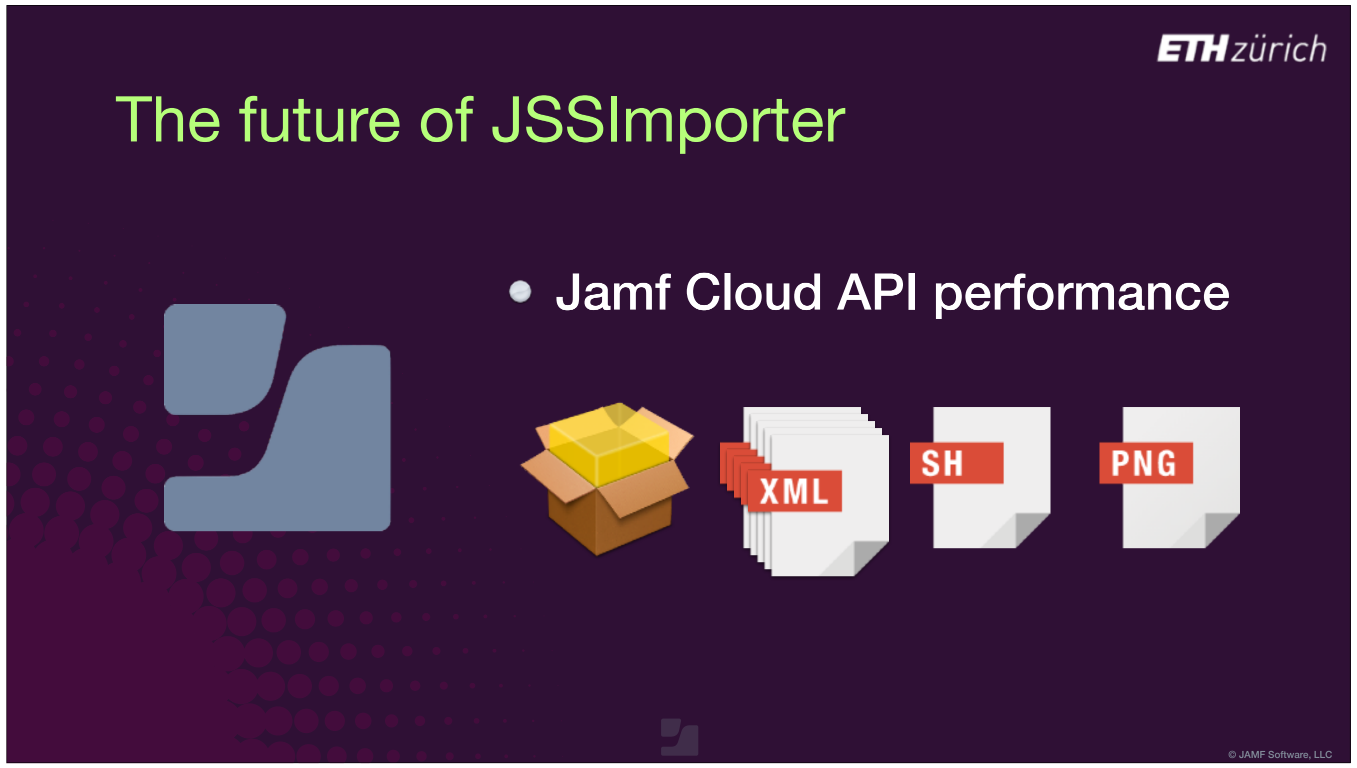
© JAMF Software, LLC

- Another problem that we have faced with the migration to Jamf Cloud is the performance of API requests.
- The way JSSImporter works is to send a series of separate API requests during the recipe run, to make all the required changes:
  - [click] Uploading the package and creating a package object in Jamf, creating any new categories, creating or updating any linked scripts and extension attributes, creating or updating the smart group, creating or updating the policy, and uploading the self service icon.
- Many of these API requests depend on each other – for instance, the policy creation request will fail, if the linked categories and smart groups don't exist.

ETH zürich

The future of JSSImporter

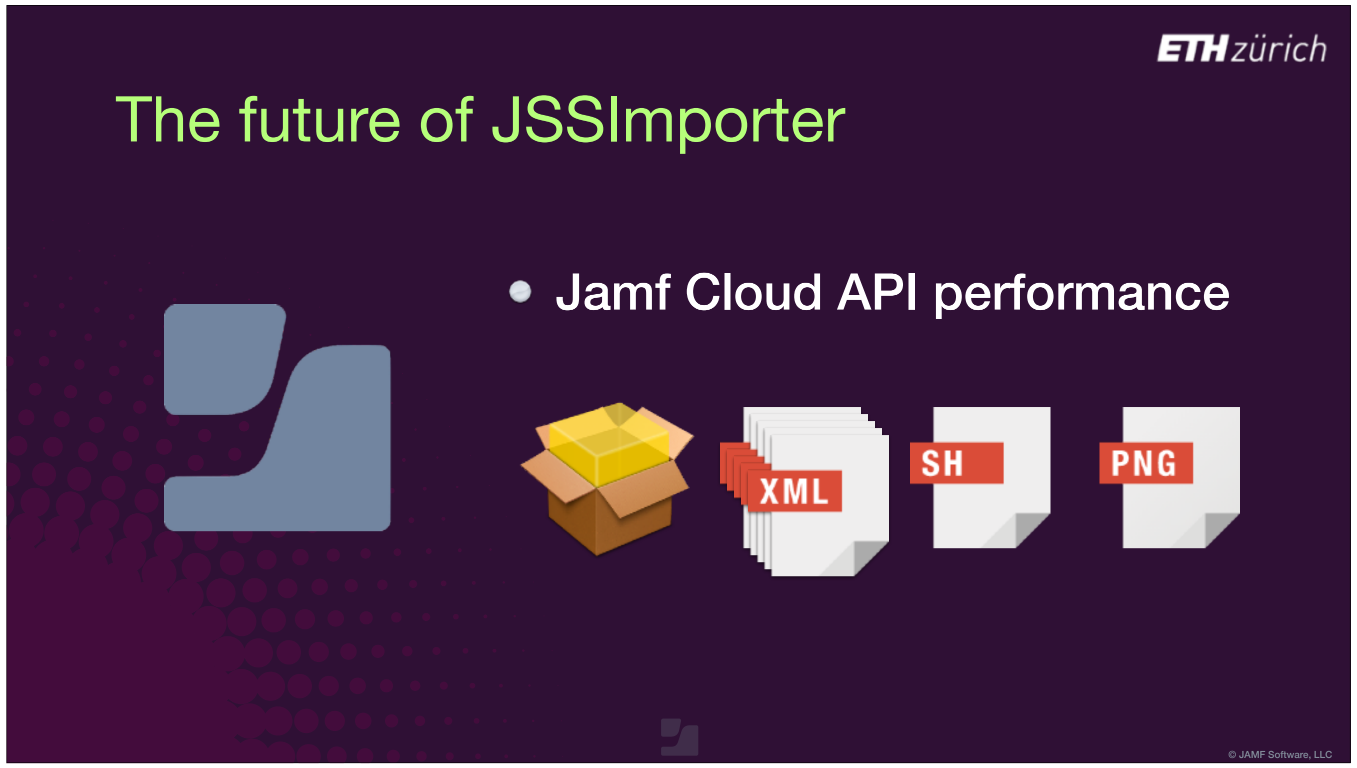
- Jamf Cloud API performance



ETH zürich

The future of JSSImporter

- Jamf Cloud API performance



The future of JSSImporter

- Jamf Cloud API performance

ETH zürich
XML
SH
PNG
© JAMF Software, LLC

- The problem is that Jamf Cloud is clustered, but there's no method for the API to hit the same node for each request in the sequence.
- So we get false conflicts and errors when we make a request to a different node before the previous request has synchronised.
- We've tried adding wait loops and checks into JSSImporter to work around this, but it's not perfect.
- [click] I've also had some great discussions about this over the last couple of days, so I'm hopeful we can solve this.

ETH zürich

The future of JSSImporter

- Jamf Cloud API performance

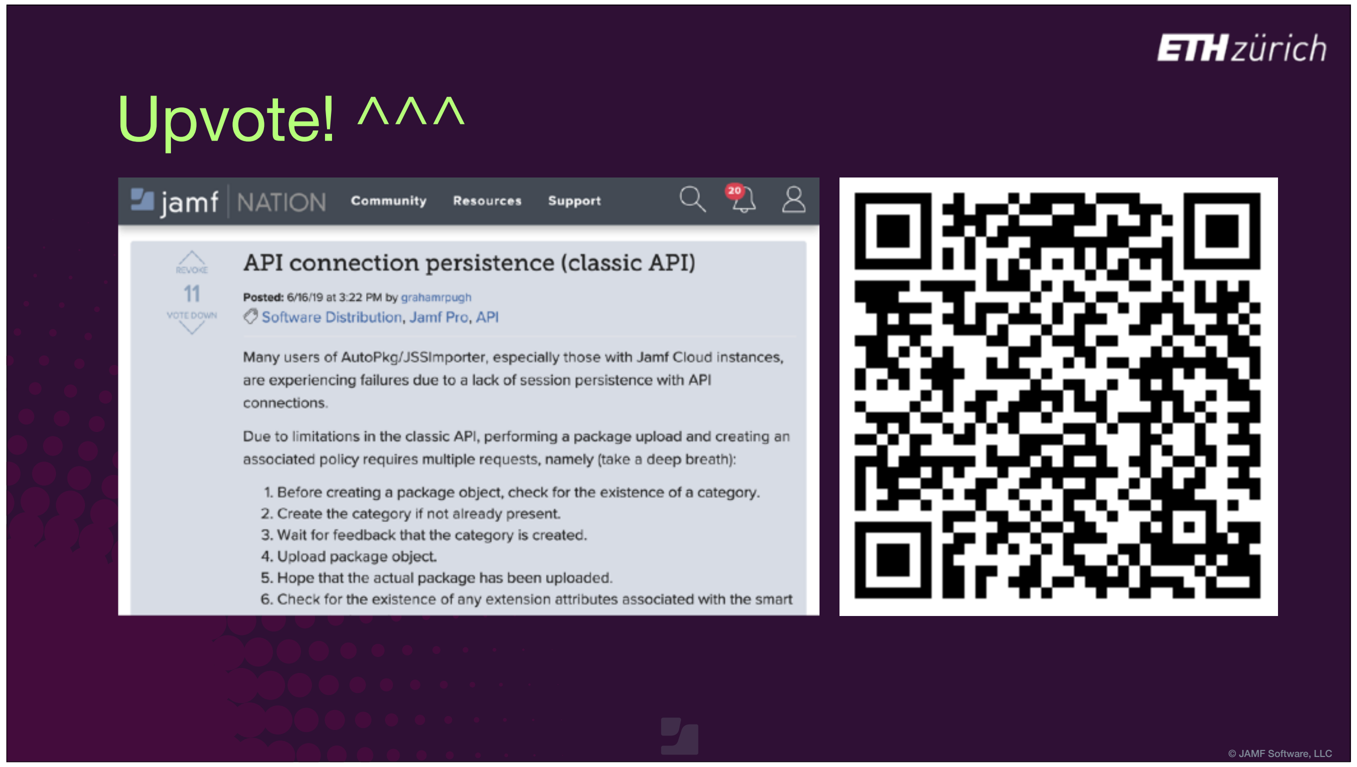
XML SH PNG

ETH zürich

The future of JSSImporter

- Jamf Cloud API performance

XML SH PNG



ETH zürich

Upvote! ^^^

jamf | NATION Community Resources Support 20

VOTE 11 VOTE DOWN

API connection persistence (classic API)

Posted: 6/16/19 at 3:22 PM by grahamrpugh

Software Distribution, Jamf Pro, API

Many users of AutoPkg/JSSImporter, especially those with Jamf Cloud instances, are experiencing failures due to a lack of session persistence with API connections.

Due to limitations in the classic API, performing a package upload and creating an associated policy requires multiple requests, namely (take a deep breath):

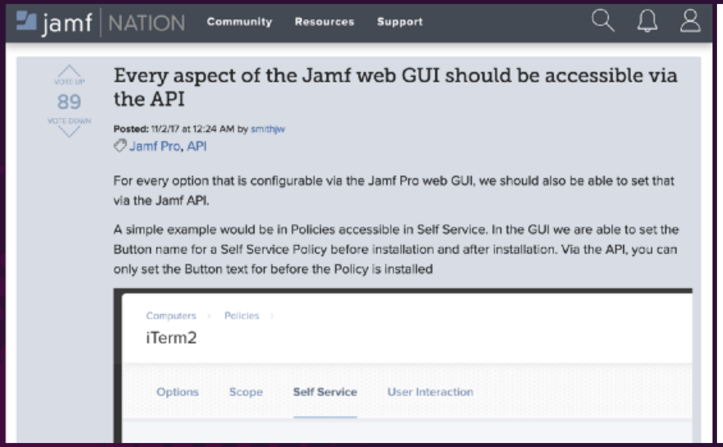
1. Before creating a package object, check for the existence of a category.
2. Create the category if not already present.
3. Wait for feedback that the category is created.
4. Upload package object.
5. Hope that the actual package has been uploaded.
6. Check for the existence of any extension attributes associated with the smart

© JAMF Software, LLC

- You can all help get this fixed too by upvoting this FR!

ETH zürich

Upvote! ^^^



Every aspect of the Jamf web GUI should be accessible via the API

Posted: 11/2/17 at 12:24 AM by smithjw

Jamf Pro, API

For every option that is configurable via the Jamf Pro web GUI, we should also be able to set that via the Jamf API.

A simple example would be in Policies accessible in Self Service. In the GUI we are able to set the Button name for a Self Service Policy before installation and after installation. Via the API, you can only set the Button text for before the Policy is installed

Computers › Policies ›
iTerm2
Options Scope Self Service User Interaction

- [slide was not shown in the session]
- And this one is from James Smith concerning getting a full-featured API.
- If you care about being able to use API requests to configure every aspect of your Jamf Pro servers, please consider this FR too!

The future of JSSImporter

ETH zürich
© JAMF Software, LLC

- The third thing that JSSImporter's future depends on is you!
- JSSImporter is an open source project which is widely used, but Shea Craig no longer involved, and there's a lot to do to keep it alive.
- Whether you work at Jamf, or you're a Jamf customer who is a keen Python scripter, or are just happy to help test JSSImporter in your environment, I'd be very happy to work with you.

ETH zürich

The future of JSSImporter

Links

grahamrpugh.com

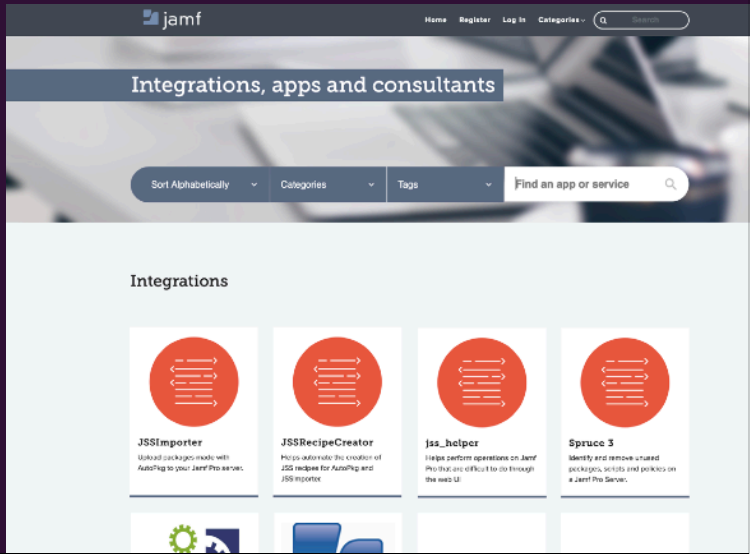
With that, thank you very much for listening!
As I said at the start, it's an essential part of our Apple device management service at ETH Zürich, and I hope it can help you out too.
Just to reiterate, I've provided links to everything I've described in this presentation, including some of the slides I didn't have time to show, in a blog post I posted today.
[click] You can also access info about AutoPkg, JSSImporter and the other tools I talked about directly from the Jamf Marketplace - thank you Charles!

- If there's time, I'm very happy to take questions now.

ETH zürich

Links

grahamrpugh.com



Integrations, apps and consultants

Integrations

JSSImporter
Upload packages made with AutoPkg to your Jamf Pro server.

JSSRecipeCreator
Helps automate the creation of JSS recipes for AutoPkg and JSSImporter.

jss_helper
Helps perform operations on Jamf Pro that are difficult to do through the web UI

Spruce 3
Identify and remove unused packages, scripts and policies on a Jamf Pro Server.

Thank you for listening!

Give us feedback by completing the 2-question session survey in the JNUC 2019 app.

UP NEXT

Jamf Pro Reporting Solutions

14:45 - 15:30