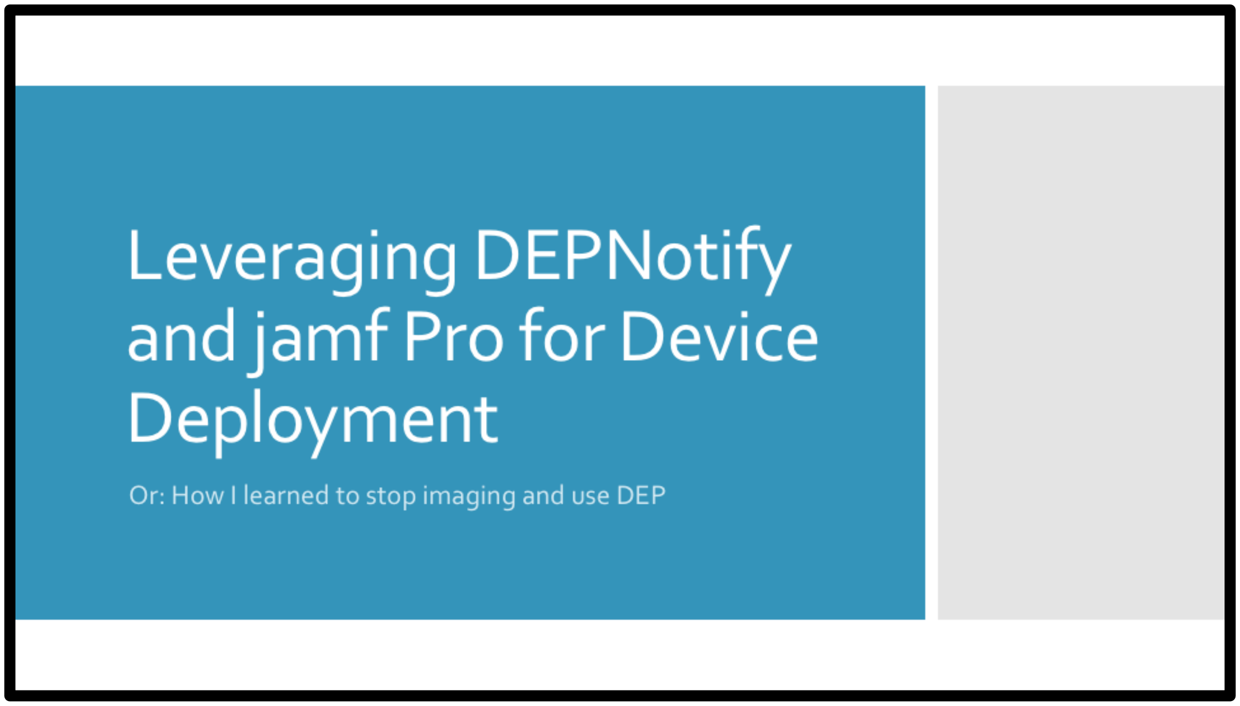
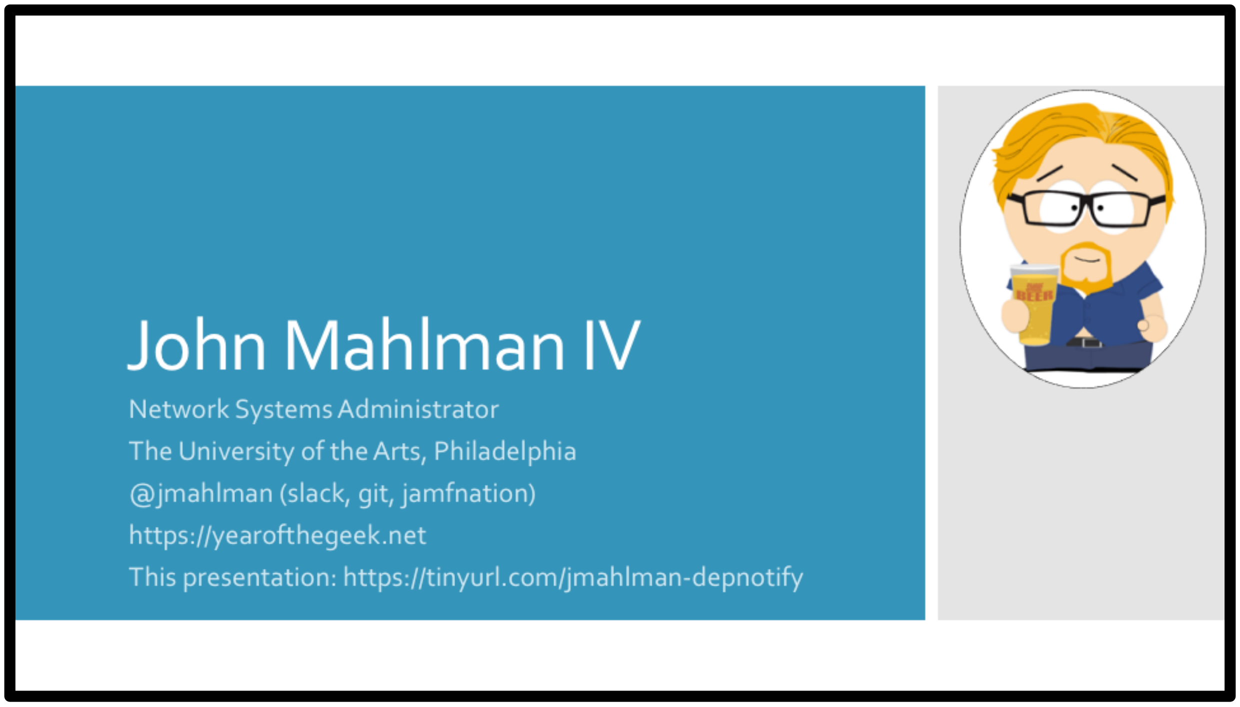


Leveraging DEPNotify and jamf Pro for Device Deployment

Or: How I learned to stop imaging and use DEP



John Mahlman IV

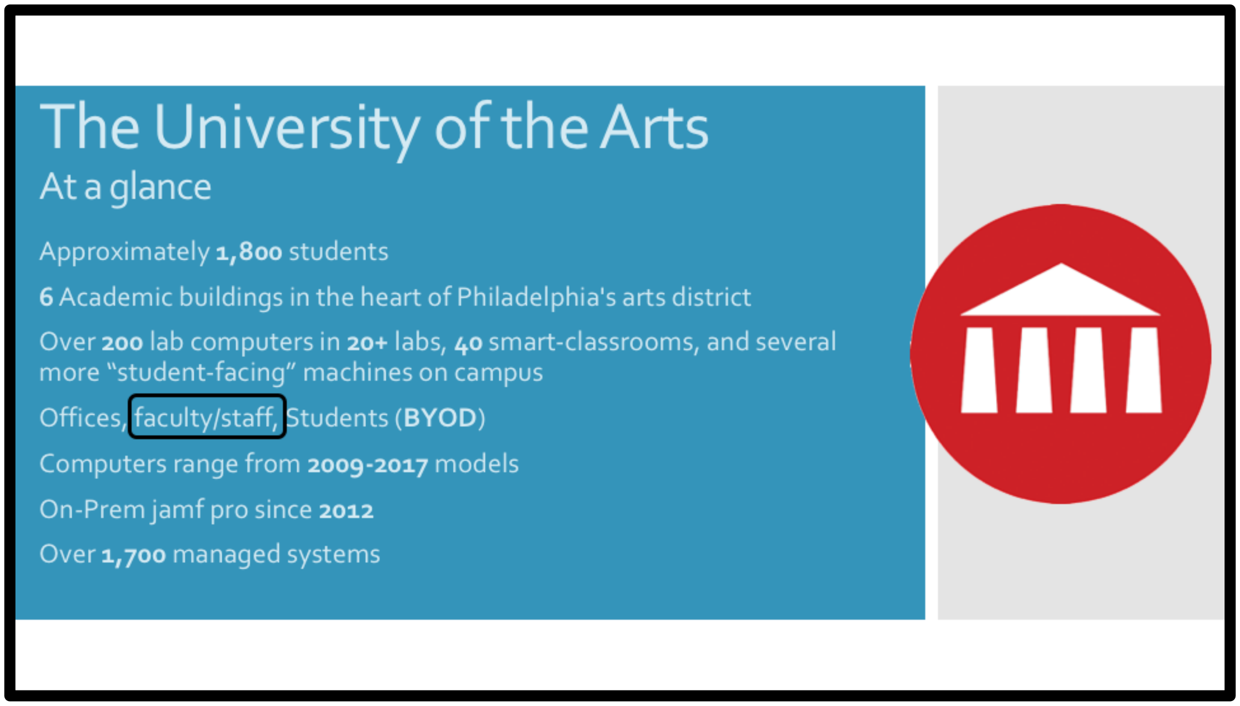
Network Systems Administrator
The University of the Arts, Philadelphia
@jmahlman (slack, git, jamfnation)
https://yearofthegeek.net
This presentation: https://tinyurl.com/jmahlman-depnotify

Just a bit about me. I'm a "Network Systems Administrator" at The University of the Arts in Philadelphia, I've been there since December 2015.

I am jmahlman on the various platforms. I have a twitter but it's a personal account and I don't really share that out too often.

And my little blog/website is there, yearfothegeek.net. I don't post often but it's basically where I go to put my processes up for future reference and hopefully to help people out. I actually decided to do this presentation because I wrote a post about this process and started to get questions from people so I figured presenting on it would be good!

_C_ This presentation can also be downloaded at the following link.



The University of the Arts

At a glance

Approximately **1,800** students

6 Academic buildings in the heart of Philadelphia's arts district

Over **200** lab computers in **20+** labs, **40** smart-classrooms, and several more "student-facing" machines on campus

Offices, faculty/staff, Students (**BYOD**)

Computers range from **2009-2017** models

On-Prem jamf pro since **2012**

Over **1,700** managed systems

As mentioned, I work for The University of the Arts in Philadelphia. Here are some stats at a glance.

_C_ We have around 1800 students _C_ spread across 6 buildings in the heart of downtown Philadelphia. We're a majority Mac campus, only a handful of windows machines…so

_C_ We have over 200 lab machines in a little over 20 labs, 40 smart-classrooms which are single systems hooked up to projectors and sound and a few other student facing systems such as studios and production suites.

_C_ We also manage office systems, faculty and staff laptops which are university provided, and finally student BYOD systems. We only supply software to students via self service, we don't do much else with those.

_C_ We have a lot of variation in systems…2009-2017 models are in use with operating systems from 10.9 to 10.13.

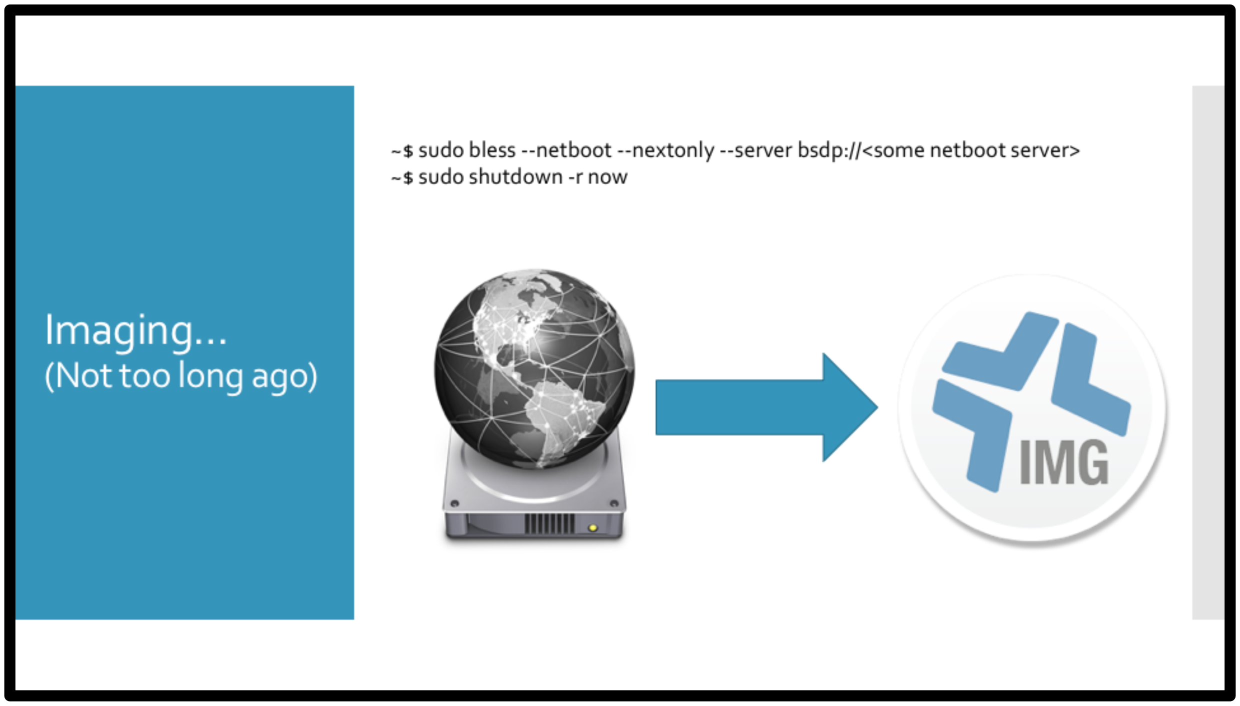
_C_ We've had on-prem jamf pro since 2012 _C_ with over 1700 systems at any time in there.

For the purpose of this presentation we’re going to focus only on _C_ faculty and staff laptops.

Let’s go back in time a few years..er..months…

Hop in, plenty of time to explain!

So, let’s go back in time a little bit… _C_



Imaging…
(Not too long ago)

```
~$ sudo bless --netboot --nextonly --server bsdp://<some netboot server>
~$ sudo shutdown -r now
```

Not too long ago..the imaging process looked like this…

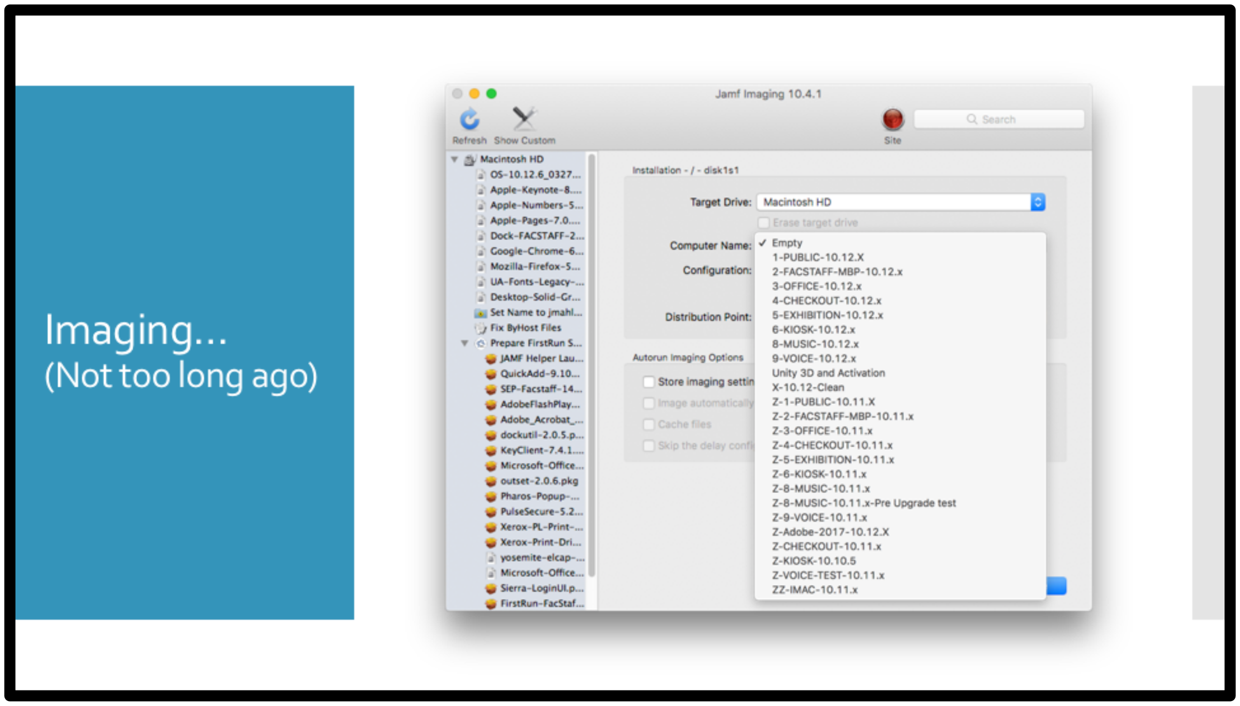
_C_ You can send out this command or something like it to a bunch of machines using ARD or jamf remote or a policy and your machine would boot to a netboot set that was on a server

_C_ then your imaging application (jamf imaging) would take over.

Imaging…
(Not too long ago)

Sit back and have a beer. And walk in the next day or whatever and your machines would be imaged.

We did this a lot actually, during summer maintenance we would scope a bunch of labs with our netboot policy, have autorun set and we didn't have to babysit anything. It also worked well for the faculty laptops…



Imaging…
(Not too long ago)

Jamf Imaging 10.4.1

Refresh Show Custom Site Search

- Macintosh HD
  - OS-10.12.6_0327…
  - Apple-Keynote-8.…
  - Apple-Numbers-5.…
  - Apple-Pages-7.0.…
  - Dock-FACSTAFF-2…
  - Google-Chrome-6…
  - Mozilla-Firefox-5…
  - UA-Fonts-Legacy-…
  - Desktop-Solid-Gr…
  - Set Name to jmahl…
  - Fix ByHost Files
  - Prepare FirstRun S…
    - JAMF Helper Lau…
    - QuickAdd-9.10…
    - SEP-Facstaff-14…
    - AdobeFlashPlay…
    - Adobe_Acrobat_…
    - dockutil-2.0.5.p…
    - KeyClient-7.4.1.…
    - Microsoft-Office…
    - outset-2.0.6.pkg
    - Pharos-Popup-…
    - PulseSecure-5.2…
    - Xerox-PL-Print-…
    - Xerox-Print-Dri…
    - yosemite-elcap-…
    - Microsoft-Office…
    - Sierra-LoginUI.p…
    - FirstRun-FacStaf…

Installation - / - disk1s1

Target Drive: Macintosh HD

Erase target drive

Computer Name:

Configuration:

Distribution Point:

✓ Empty
1-PUBLIC-10.12.X
2-FACSTAFF-MBP-10.12.x
3-OFFICE-10.12.x
4-CHECKOUT-10.12.x
5-EXHIBITION-10.12.x
6-KIOSK-10.12.x
8-MUSIC-10.12.x
9-VOICE-10.12.x
Unity 3D and Activation
X-10.12-Clean
Z-1-PUBLIC-10.11.X
Z-2-FACSTAFF-MBP-10.11.x
Z-3-OFFICE-10.11.x
Z-4-CHECKOUT-10.11.x
Z-5-EXHIBITION-10.11.x
Z-6-KIOSK-10.11.x
Z-8-MUSIC-10.11.x
Z-8-MUSIC-10.11.x-Pre Upgrade test
Z-9-VOICE-10.11.x
Z-Adobe-2017-10.12.X
Z-CHECKOUT-10.11.x
Z-KIOSK-10.10.5
Z-VOICE-TEST-10.11.x
ZZ-IMAC-10.11.x

Autorun Imaging Options

Store imaging settin

Image automatically

Cache files

Skip the delay confi

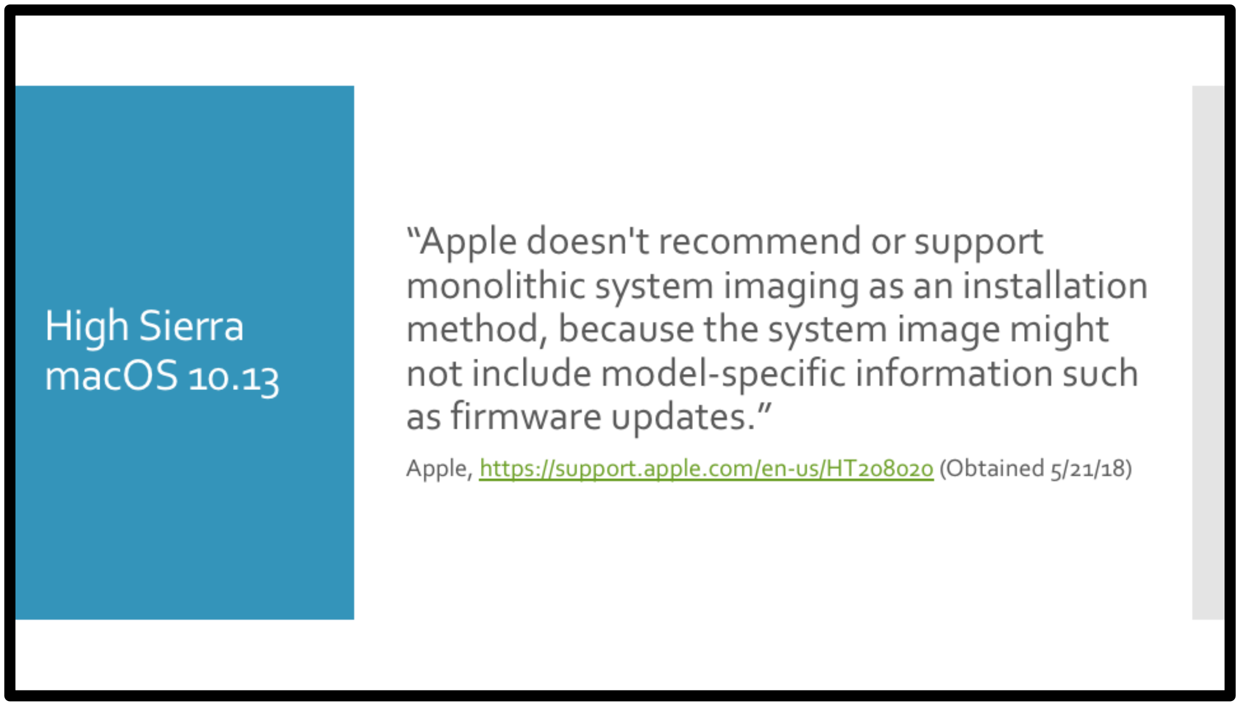
Imaging would load the configuration based on the machine record or pre-stage, or our helpdesk manually chose one and it just worked.

And things were good.

And then *it* happened..

You all know what I'm talking about…

And then…it happened. _C_ High Sierra….



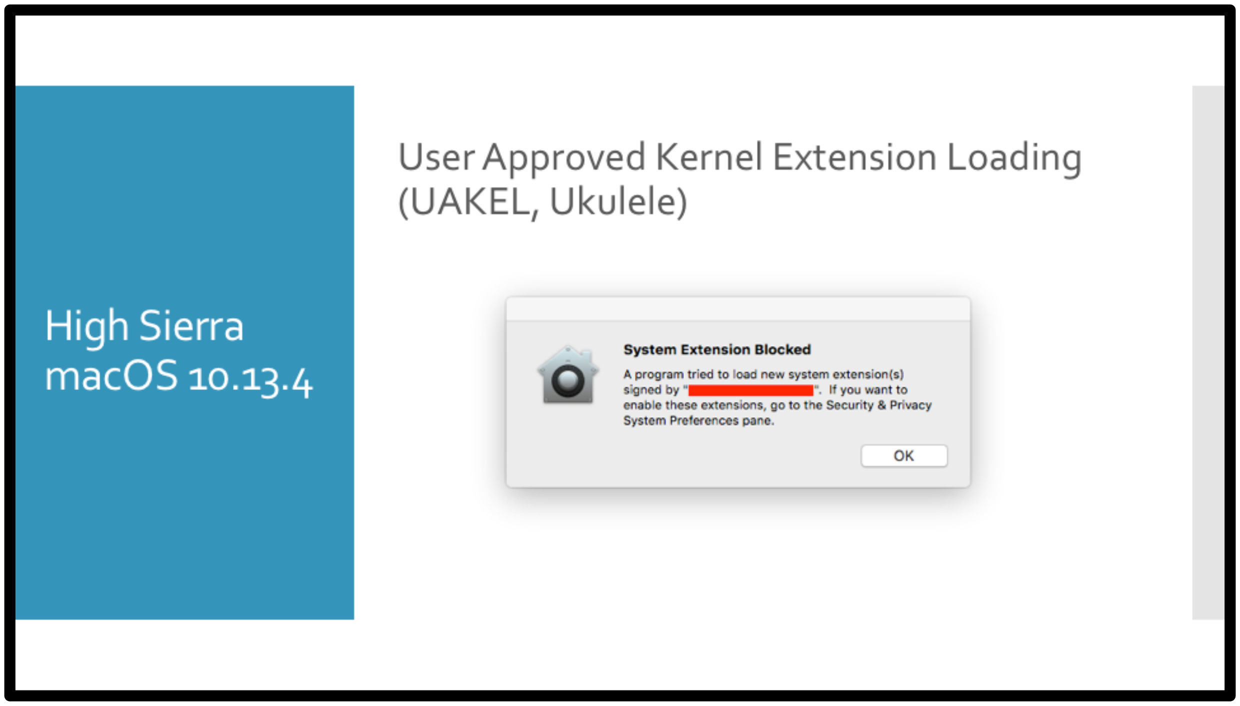
High Sierra macOS 10.13

"Apple doesn't recommend or support monolithic system imaging as an installation method, because the system image might not include model-specific information such as firmware updates."

Apple, https://support.apple.com/en-us/HT208020 (Obtained 5/21/18)

Apple gave us this wonderful update.

But I mean, just because it's not recommended doesn't mean it wouldn't work..or..



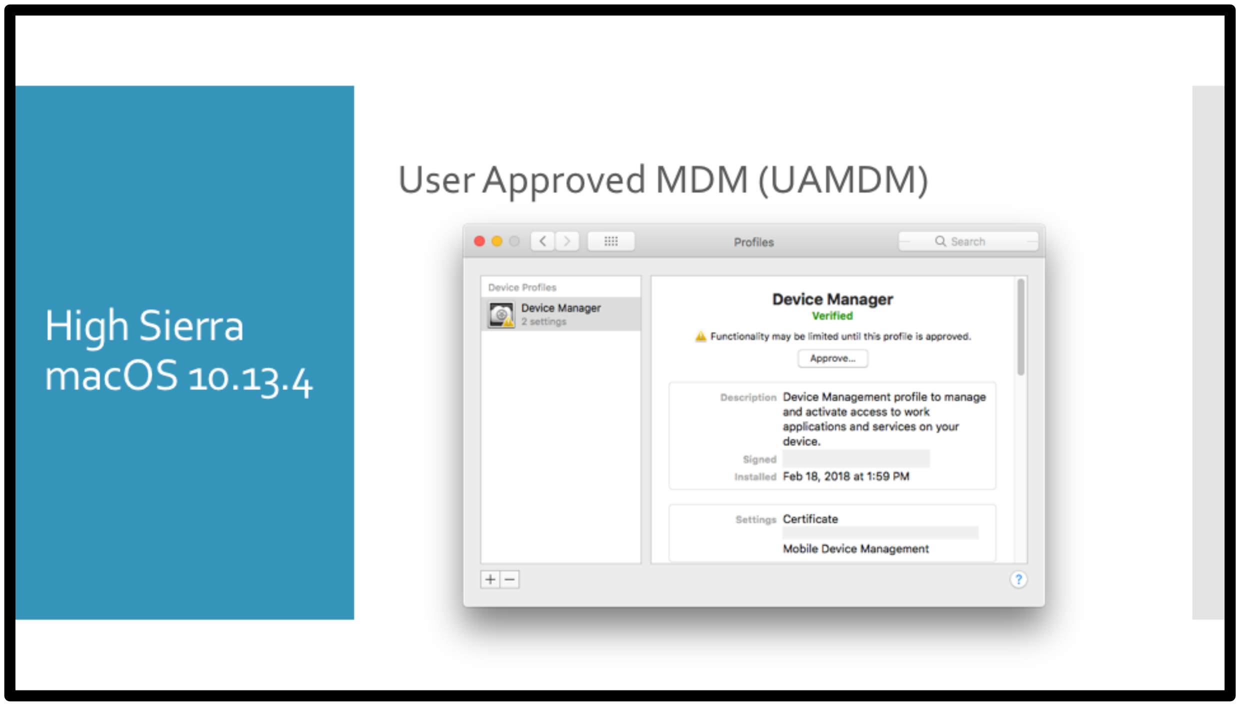
High Sierra
macOS 10.13.4

User Approved Kernel Extension Loading (UAKEL, Ukulele)

System Extension Blocked

A program tried to load new system extension(s) signed by "[redacted]". If you want to enable these extensions, go to the Security & Privacy System Preferences pane.

AH, yeah…10.13.14 gave us UAKEL.



High Sierra macOS 10.13.4

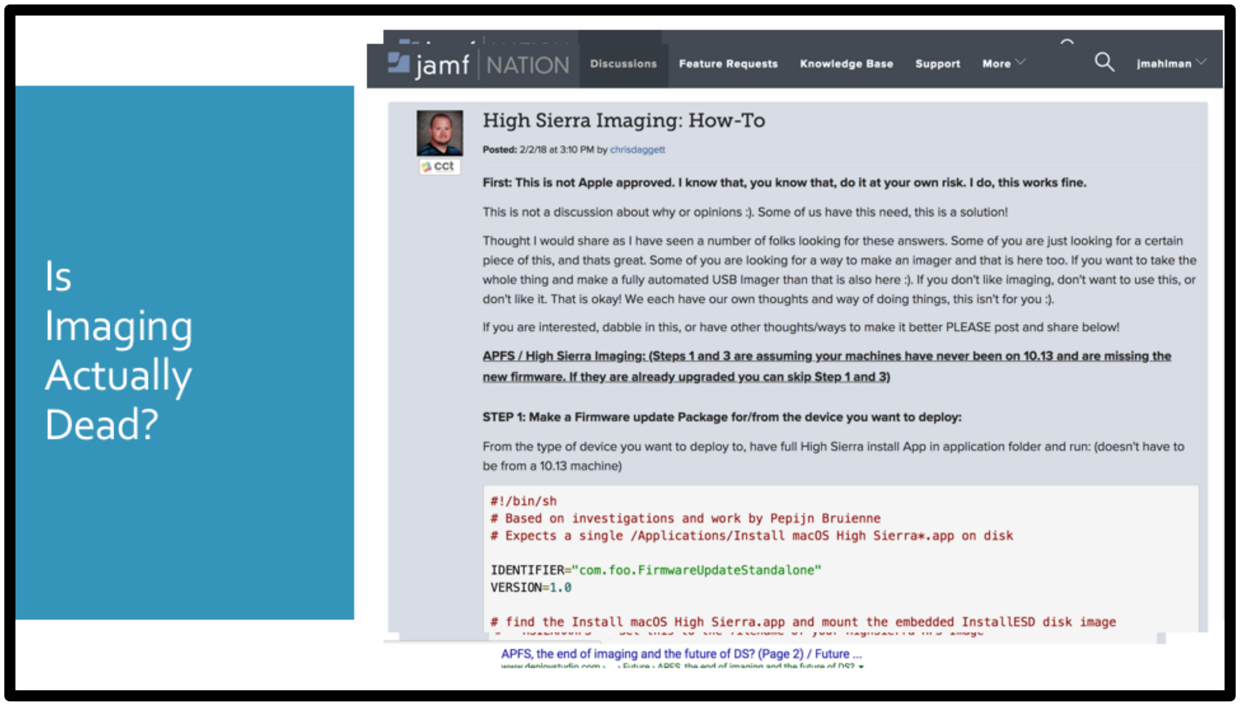
User Approved MDM (UAMDM)

Device Manager
Verified
Functionality may be limited until this profile is approved.

Description: Device Management profile to manage and activate access to work applications and services on your device.
Signed
Installed: Feb 18, 2018 at 1:59 PM
Settings: Certificate
Mobile Device Management

And then UAMDM.

These essentially killed any automated imaging..so the question that’s been asked for months now _C_



Is Imaging Actually Dead?

High Sierra Imaging: How-To

Posted: 2/2/18 at 3:10 PM by chrisdaggett

First: This is not Apple approved. I know that, you know that, do it at your own risk. I do, this works fine.

This is not a discussion about why or opinions :). Some of us have this need, this is a solution!

Thought I would share as I have seen a number of folks looking for these answers. Some of you are just looking for a certain piece of this, and thats great. Some of you are looking for a way to make an imager and that is here too. If you want to take the whole thing and make a fully automated USB Imager than that is also here :). If you don't like imaging, don't want to use this, or don't like it. That is okay! We each have our own thoughts and way of doing things, this isn't for you :).

If you are interested, dabble in this, or have other thoughts/ways to make it better PLEASE post and share below!

APFS / High Sierra Imaging: (Steps 1 and 3 are assuming your machines have never been on 10.13 and are missing the new firmware. If they are already upgraded you can skip Step 1 and 3)

STEP 1: Make a Firmware update Package for/from the device you want to deploy:

From the type of device you want to deploy to, have full High Sierra install App in application folder and run: (doesn't have to be from a 10.13 machine)

```
#!/bin/sh
# Based on investigations and work by Pepijn Bruienne
# Expects a single /Applications/Install macOS High Sierra*.app on disk

IDENTIFIER="com.foo.FirmwareUpdateStandalone"
VERSION=1.0

# find the Install macOS High Sierra.app and mount the embedded InstallESD disk image
```

APFS, the end of imaging and the future of DS? (Page 2) / Future ...

Is imaging actually dead?

Let's check google. _C_. Wow, that's a lot of stuff..let's click some of them.

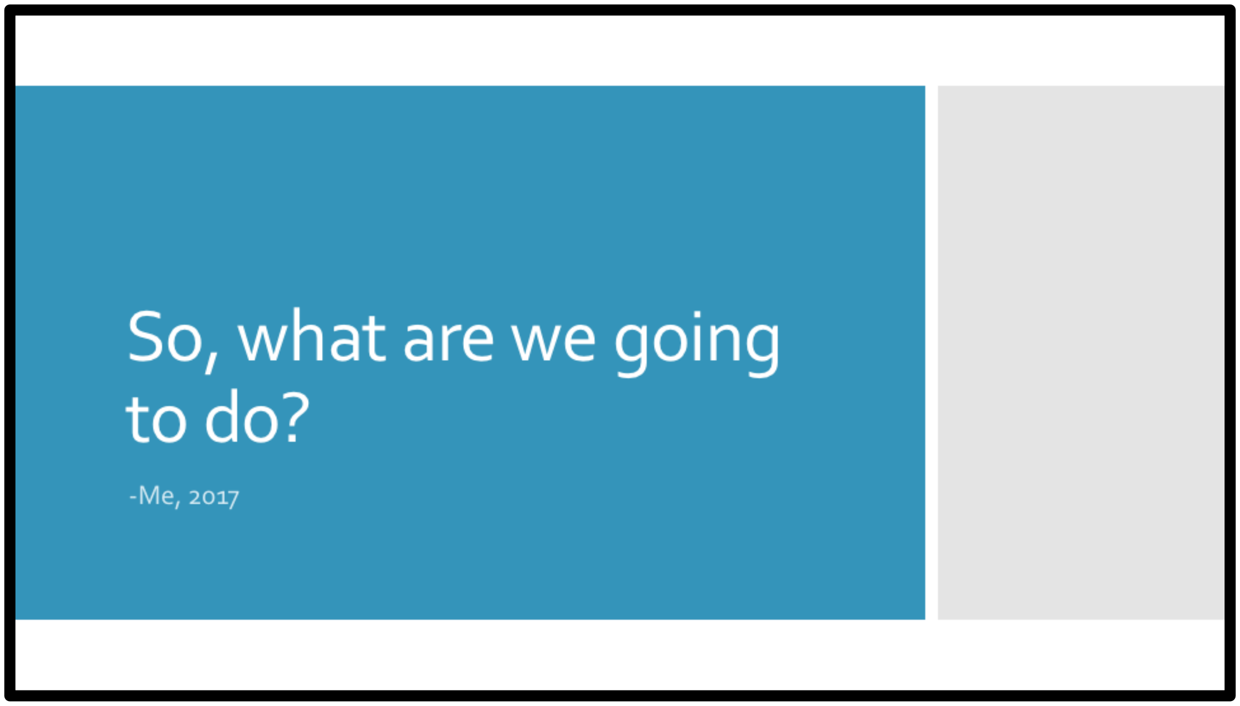
Or buddy **Armin Briegel _C_** Oh, yeah.. Okay, how about Jamf Nation _C_

Yeah..posts about people basically learning how to deploy again. _C_

So yeah..

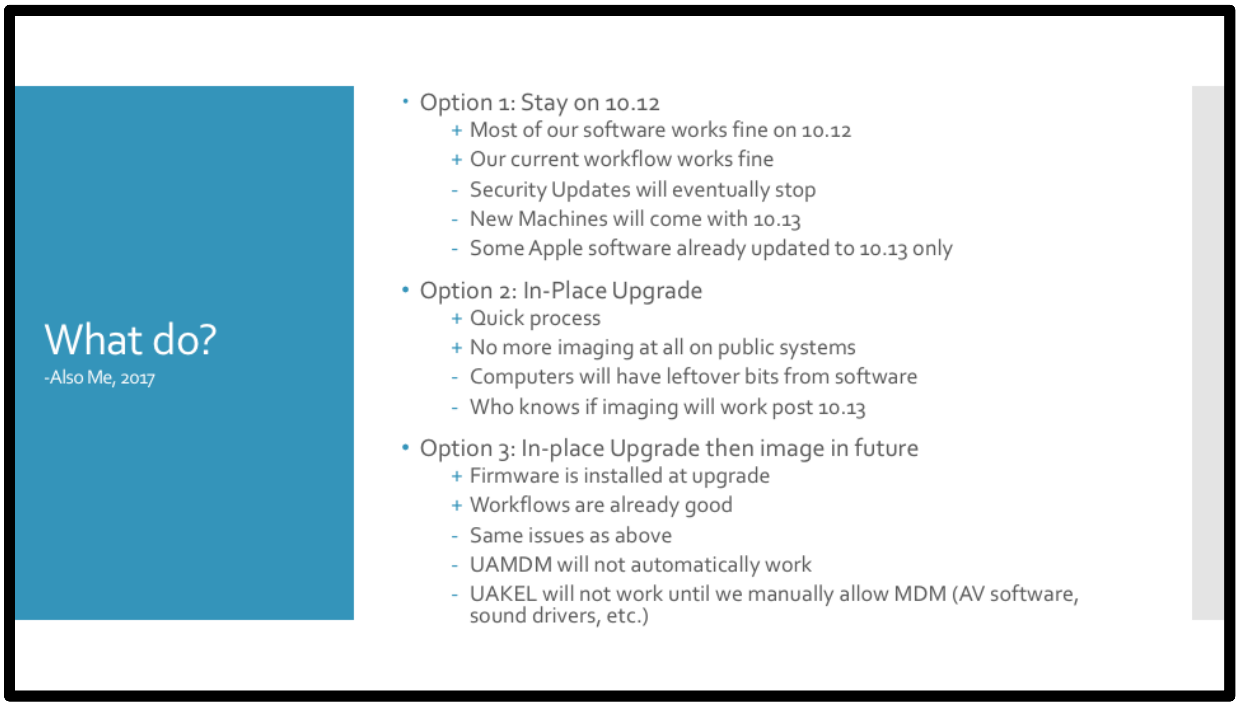
So yeah, it’s dead , Jim_C_

Mostly. Because let’s be honest, it still works, but that may not last for long.. _C_



So, what are we going to do?

-Me, 2017



What do?

-Also Me, 2017

- Option 1: Stay on 10.12
  - + Most of our software works fine on 10.12
  - + Our current workflow works fine
  - - Security Updates will eventually stop
  - - New Machines will come with 10.13
  - - Some Apple software already updated to 10.13 only
- Option 2: In-Place Upgrade
  - + Quick process
  - + No more imaging at all on public systems
  - - Computers will have leftover bits from software
  - - Who knows if imaging will work post 10.13
- Option 3: In-place Upgrade then image in future
  - + Firmware is installed at upgrade
  - + Workflows are already good
  - - Same issues as above
  - - UAMDM will not automatically work
  - - UAKEL will not work until we manually allow MDM (AV software, sound drivers, etc.)

Well, we looked at a bunch of options.

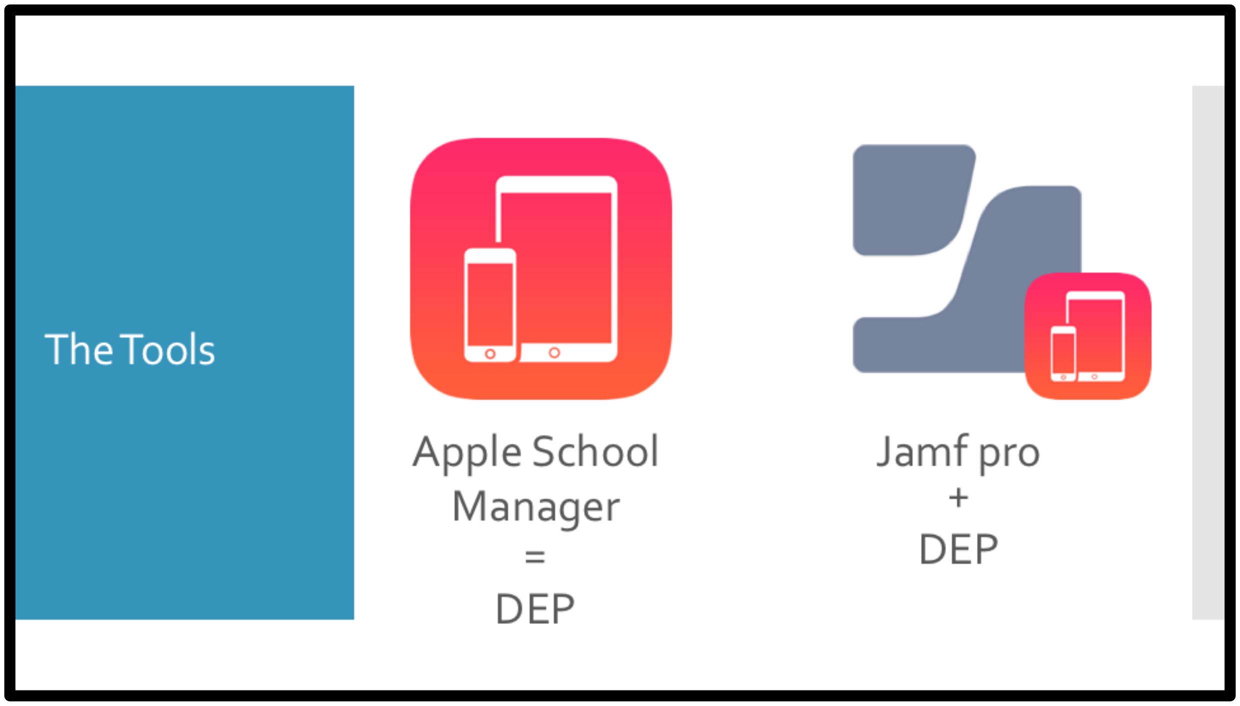
Our first option was to just stay on 10.12. **<READ LIST>** This would be fine for us in the short term but not in the long term at all. _C_

Our next option was to do an in-place upgrade. Apple gives us the tools, right? Let's use them…so the pros and cons again. **<READ LIST>** This one was looking good for us but we still didn't like those cons at all. _C_

Option 3 was sort of a hybrid solution and I'll be honest with you, this is the process we're using as a very quick stop-gap for our labs. Because again, imaging STILL works on 10.13..but we would run into the same issues again plus the UAMDM and UAKEL issues, unless you image to 10.13.3 then upgrade..but for a long term solution, that won't work.

Really, what are we going to do?

-Also also Me, 2018



The Tools

Apple School Manager = DEP

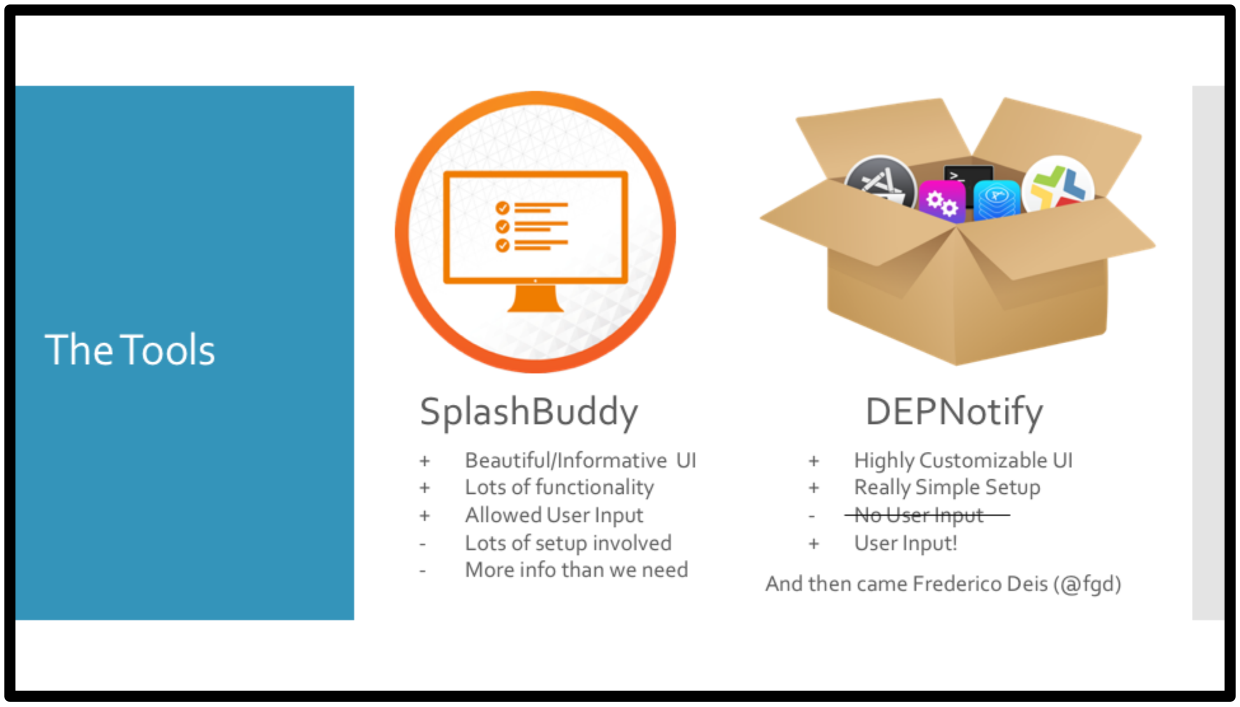
Jamf pro + DEP

Let's look at the tools we have.

We have Apple School manager _C_ which is just DEP for schools.

And we have jamf pro _C_ which works with DEP

Okay, great, now how to we put these together and make is simple and a nice user experience? Well..there are apps for that! And we took a look at two of them. _C_



The Tools

SplashBuddy

+ Beautiful/Informative UI
+ Lots of functionality
+ Allowed User Input
- Lots of setup involved
- More info than we need

DEPNotify

+ Highly Customizable UI
+ Really Simple Setup
- ~~No User Input~~
+ User Input!

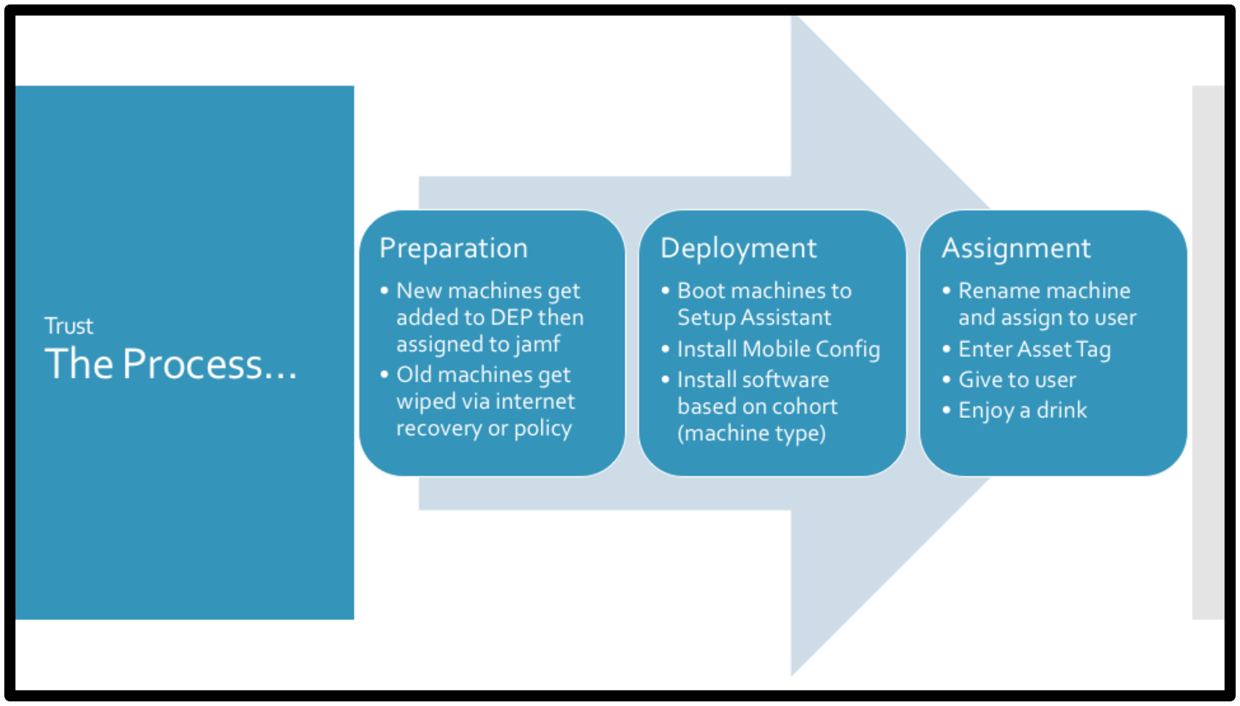
And then came Frederico Deis (@fgd)

The first one was SplashBuddy and the other was the new kid _C_ DEPNotify.

Splashbuddy was very enticing _C_ Looking at the pros and cons **<READ LIST>**

_C_ And now DEPnotify. **<READ LIST>** Now, that last one was pretty important to us, we wanted user input so we can name machines and add asset information..so we were really going to go for splash buddy. _C_

And then came Frederico Deis (FGD on slack). He posted in the depnotify channel that he was looking for people to test his forked build that offered user input! So I downloaded, tested it and figured out how to use it and it worked really well! _C_ So taking that "con" away..I had my decision.



Trust

The Process…

Preparation

- New machines get added to DEP then assigned to jamf
- Old machines get wiped via internet recovery or policy

Deployment

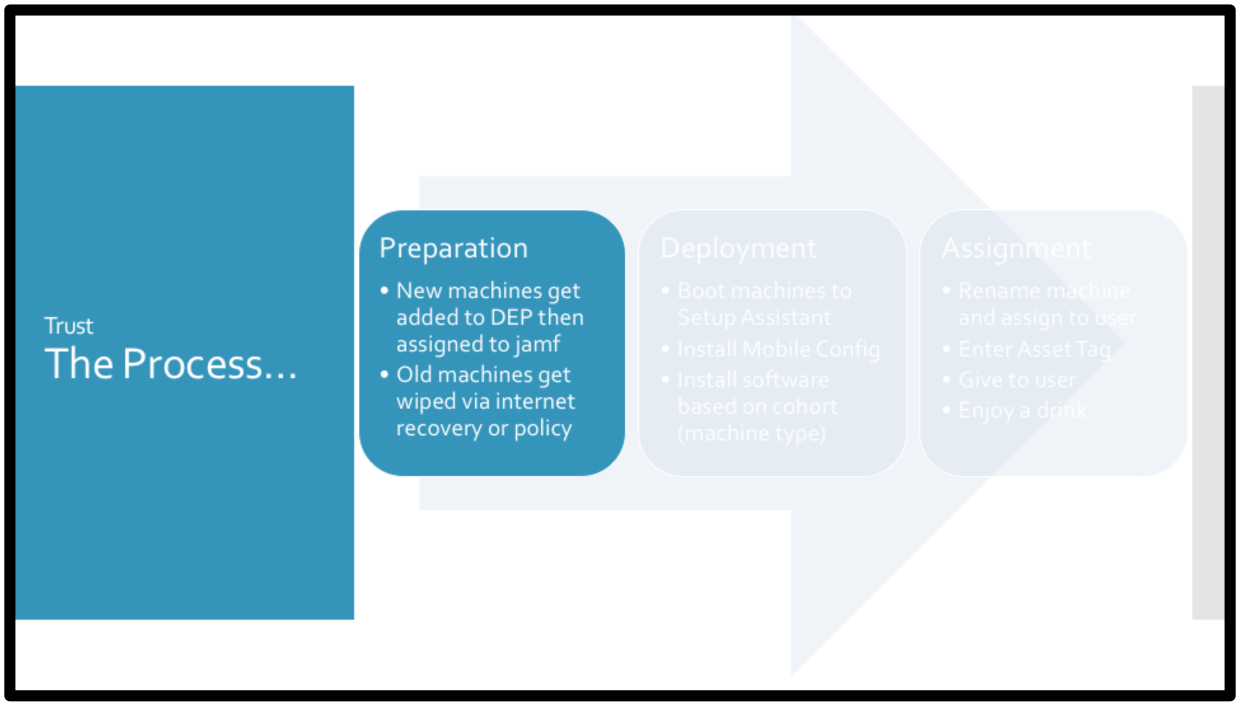
- Boot machines to Setup Assistant
- Install Mobile Config
- Install software based on cohort (machine type)

Assignment

- Rename machine and assign to user
- Enter Asset Tag
- Give to user
- Enjoy a drink

So here was the simplified process we had in place basically.

Get new machines and prep them with the base OS with either DEP for a new machine or internet recovery for a reused one. Boot machines to setup and install the software that was needed and when the machine was going to be assigned we would rename it and enter asset information then give it to the user. Of course, a cold beverage at the end.



Trust

The Process…

Preparation

- New machines get added to DEP then assigned to jamf
- Old machines get wiped via internet recovery or policy

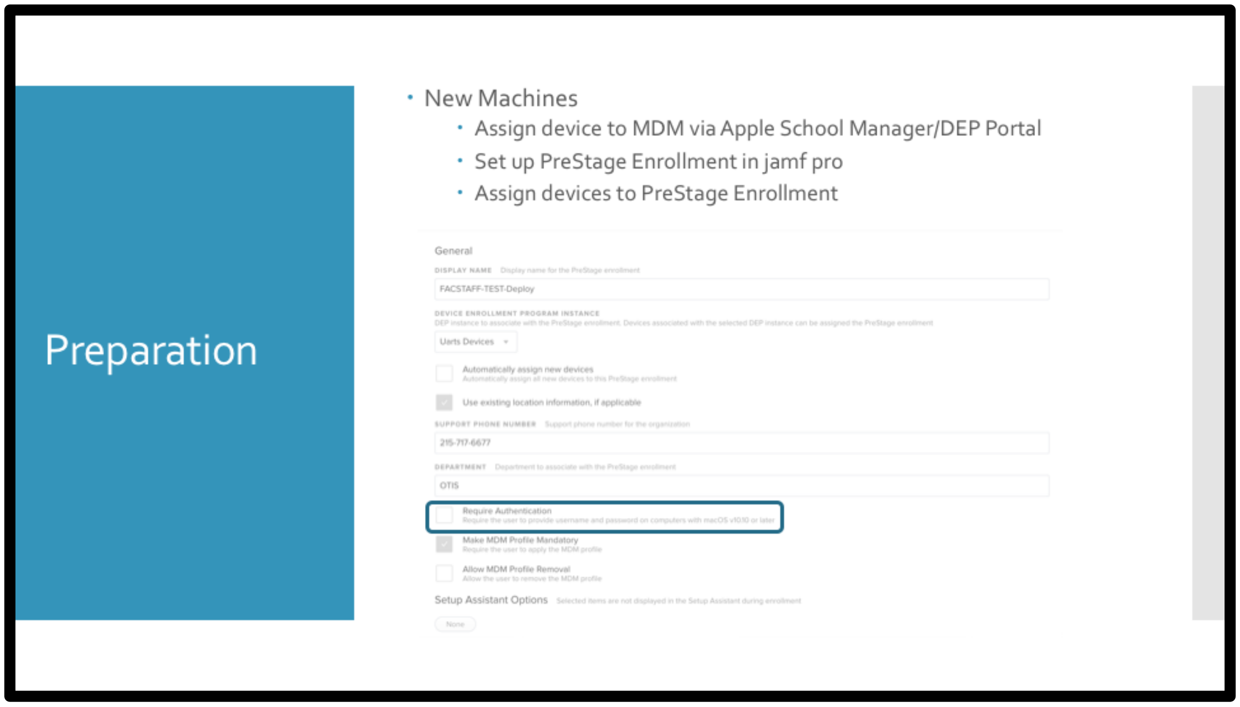
Deployment

- Boot machines to Setup Assistant
- Install Mobile Config
- Install software based on cohort (machine type)

Assignment

- Rename machine and assign to user
- Enter Asset Tag
- Give to user
- Enjoy a drink

Let's take a look at the preparation.



Preparation

- New Machines
  - Assign device to MDM via Apple School Manager/DEP Portal
  - Set up PreStage Enrollment in jamf pro
  - Assign devices to PreStage Enrollment

General

DISPLAY NAME Display name for the PreStage enrollment

FACSTAFF-TEST-Deploy

DEVICE ENROLLMENT PROGRAM INSTANCE
DEP instance to associate with the PreStage enrollment. Devices associated with the selected DEP instance can be assigned the PreStage enrollment

Uarts Devices

- [ ] Automatically assign new devices
  Automatically assign all new devices to this PreStage enrollment
- [✓] Use existing location information, if applicable

SUPPORT PHONE NUMBER Support phone number for the organization

215-717-6677

DEPARTMENT Department to associate with the PreStage enrollment

OTIS

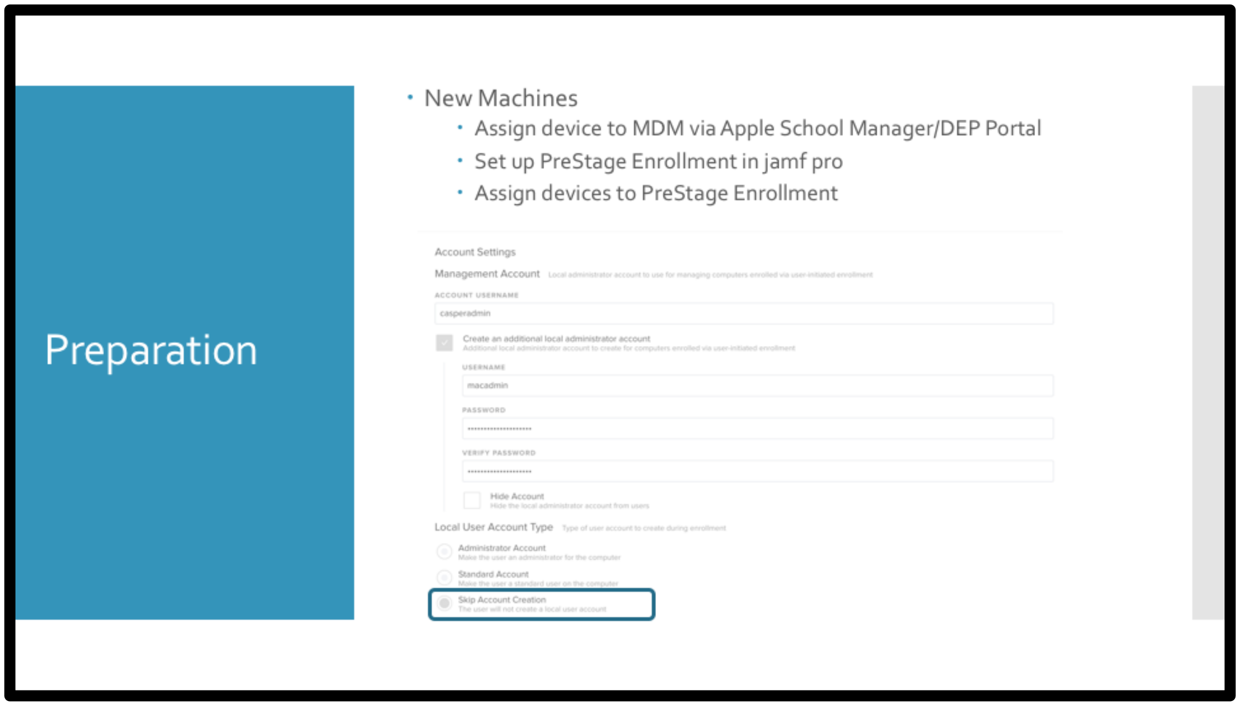
- [ ] Require Authentication
  Require the user to provide username and password on computers with macOS v10.10 or later
- [✓] Make MDM Profile Mandatory
  Require the user to apply the MDM profile
- [ ] Allow MDM Profile Removal
  Allow the user to remove the MDM profile

Setup Assistant Options Selected items are not displayed in the Setup Assistant during enrollment

None

New machines would get assigned in our server via Apple School Manager (DEP). Make sure prestage is configured then assign the machine to the correct pre-stage setup.

Note _C_ the checkbox for "require authentication" is unchecked. This is because **we** want our techs to setup the machine and if you require authentication, that machine gets assigned to that user and we didn't want that.



Preparation

- New Machines
  - Assign device to MDM via Apple School Manager/DEP Portal
  - Set up PreStage Enrollment in jamf pro
  - Assign devices to PreStage Enrollment

Account Settings

Management Account — Local administrator account to use for managing computers enrolled via user-initiated enrollment

ACCOUNT USERNAME: casperadmin

[✓] Create an additional local administrator account — Additional local administrator account to create for computers enrolled via user-initiated enrollment

USERNAME: macadmin

PASSWORD: ••••••••••••••••••••

VERIFY PASSWORD: ••••••••••••••••••••

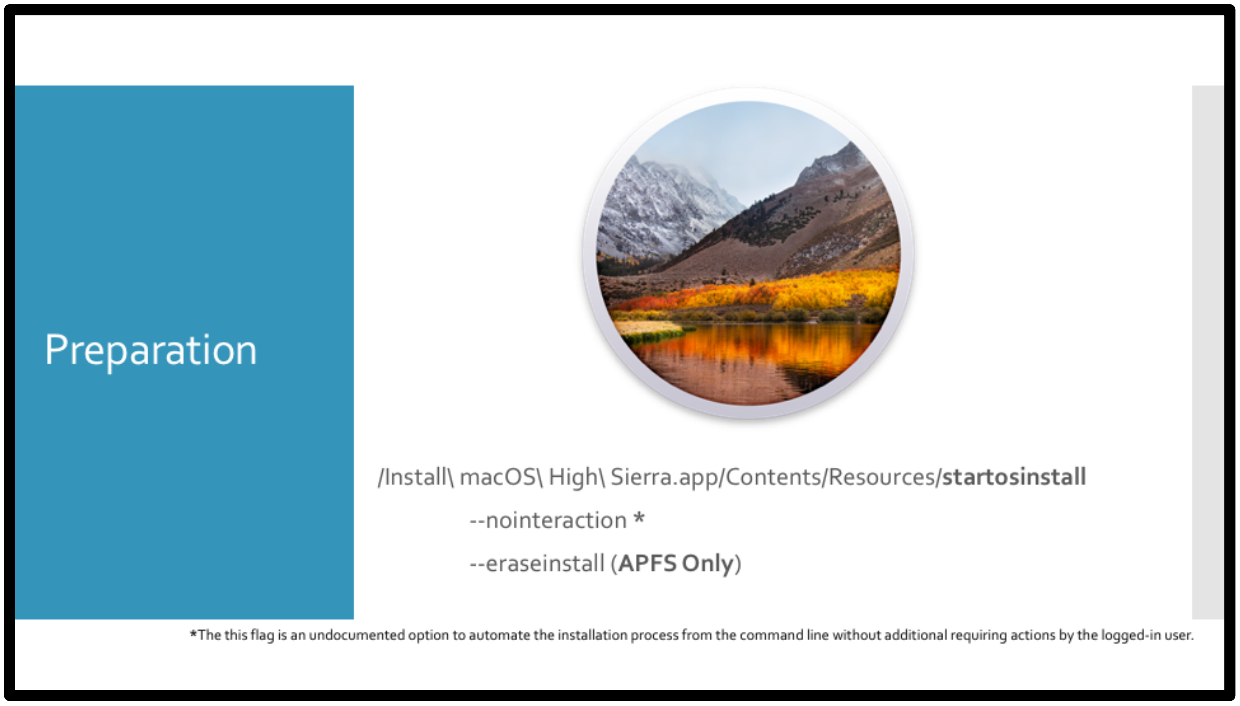
[ ] Hide Account — Hide the local administrator account from users

Local User Account Type — Type of user account to create during enrollment

- Administrator Account — Make the user an administrator for the computer
- Standard Account — Make the user a standard user on the computer
- [✓ Skip Account Creation] — The user will not create a local user account

The next setting was to create our local admin account, macadmin. That is an account that we have on all machines just in case the user forgets their password or some other reason.

Note _C_ that we skip account creation. We do this again so the machine isn't completely locked to a user until we're ready.



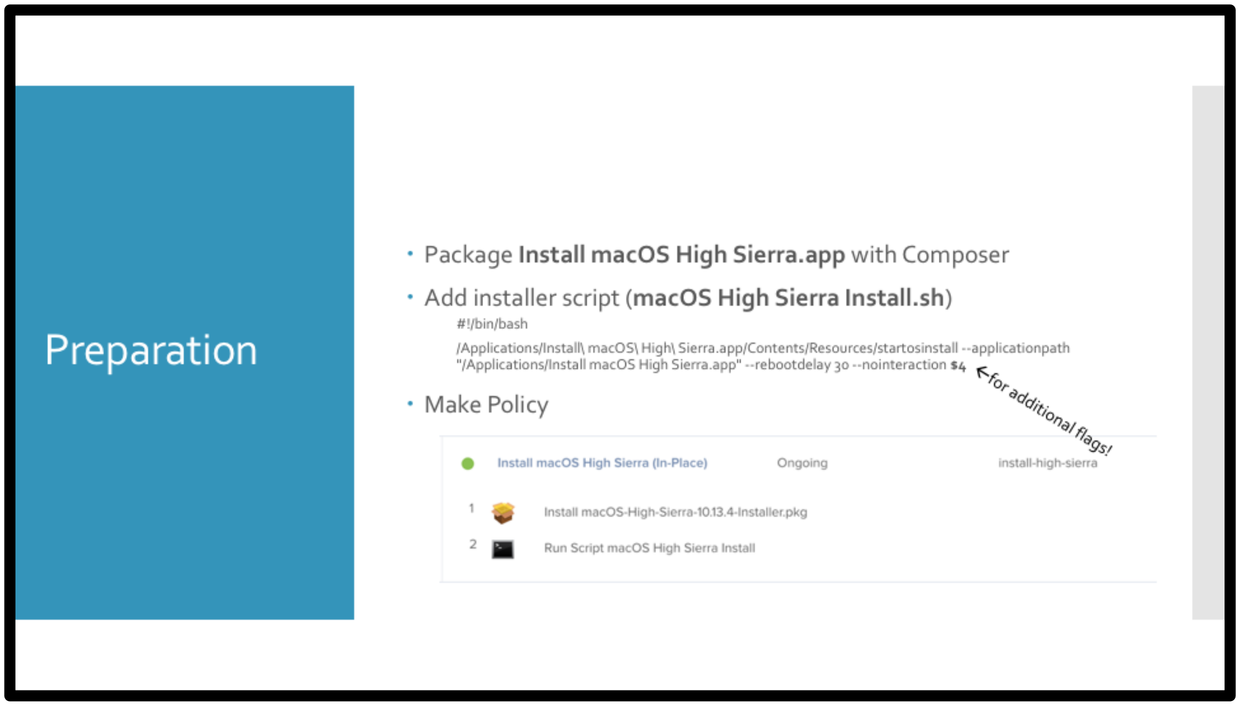
Preparation

```
/Install\ macOS\ High\ Sierra.app/Contents/Resources/startosinstall
        --nointeraction *
        --eraseinstall (APFS Only)
```

*The this flag is an undocumented option to automate the installation process from the command line without additional requiring actions by the logged-in user.

Since this presentation really focuses on our faculty and staff machines, we typically wouldn't do an automated or remote install, but I figured that it's important to know about them.

For machines already in the jamf server we can utilize the start os install binary from the High Sierra installer with a policy. There are some useful flags available to us, these are two that we like using if we can.



Preparation

- Package **Install macOS High Sierra.app** with Composer
- Add installer script (**macOS High Sierra Install.sh**)

```
#!/bin/bash
/Applications/Install\ macOS\ High\ Sierra.app/Contents/Resources/startosinstall --applicationpath
"/Applications/Install macOS High Sierra.app" --rebootdelay 30 --nointeraction $4
```

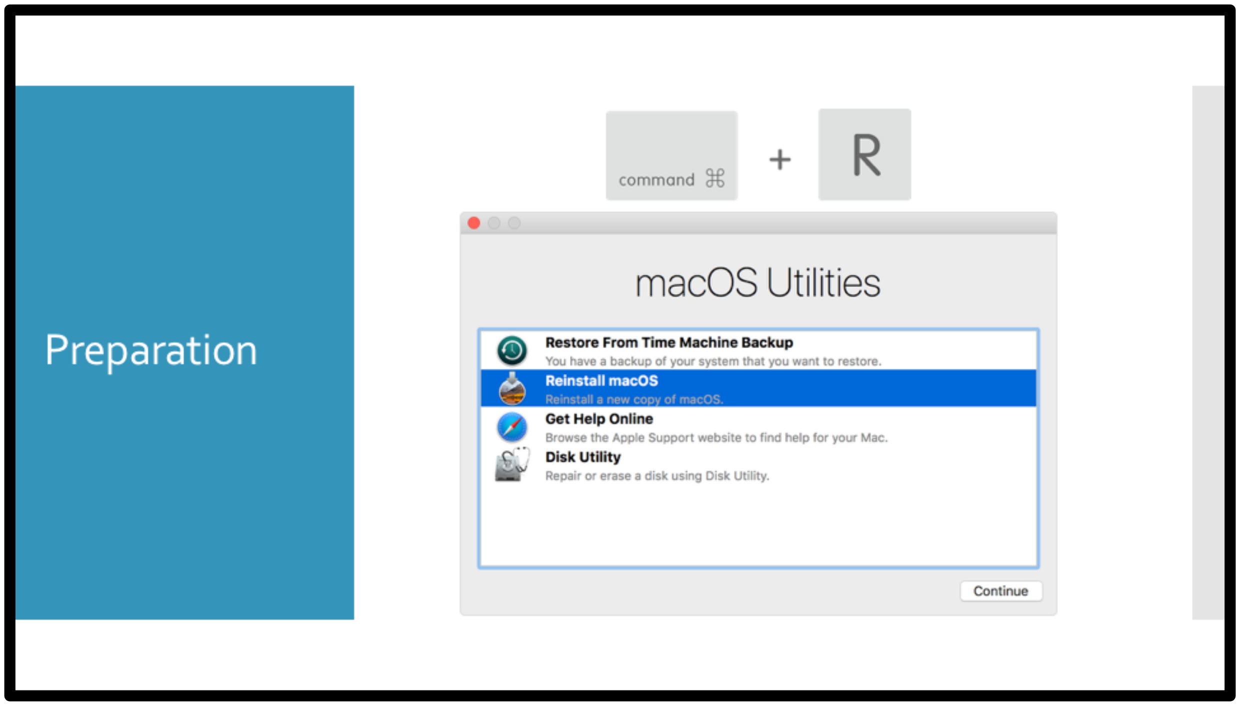
- Make Policy

```
●  Install macOS High Sierra (In-Place)     Ongoing     install-high-sierra

1  Install macOS-High-Sierra-10.13.4-Installer.pkg
2  Run Script macOS High Sierra Install
```

First you push out the installer app then run a script that will take care of the install process for us with start os install.

_C_ Note the $4 flag, we can use that for additional flags such as erase install or convert to apfs.

This is nice because if you have machines that are APFS compatible, you can make a self service policy for your techs to wipe the drive and re-install a clean OS! The example above is just using a custom trigger.



Preparation

command ⌘ + R

macOS Utilities

Restore From Time Machine Backup
You have a backup of your system that you want to restore.

Reinstall macOS
Reinstall a new copy of macOS.

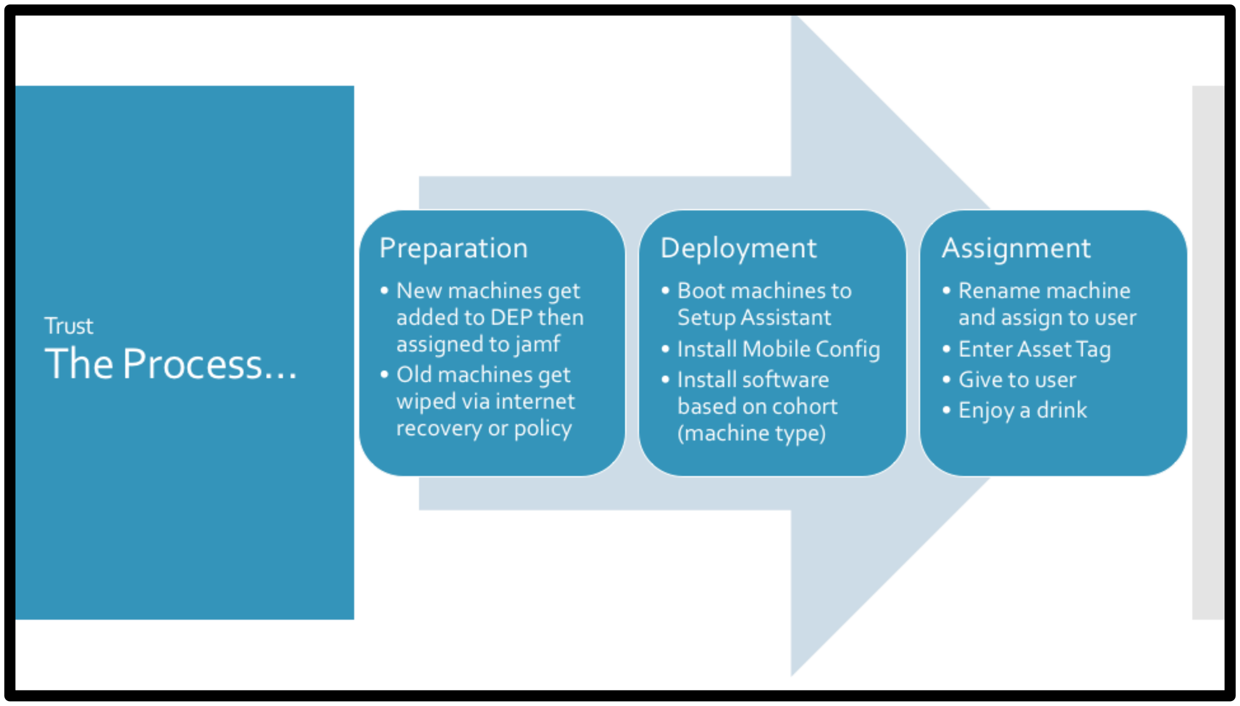
Get Help Online
Browse the Apple Support website to find help for your Mac.

Disk Utility
Repair or erase a disk using Disk Utility.

Continue

For machines that are either not in the jamf server or you cannot use the "erase install" command you should just boot to internet recovery with good ol' command R.

_C_ which of course brings up the utilities and you can go from there.



Trust

The Process…

Preparation

- New machines get added to DEP then assigned to jamf
- Old machines get wiped via internet recovery or policy

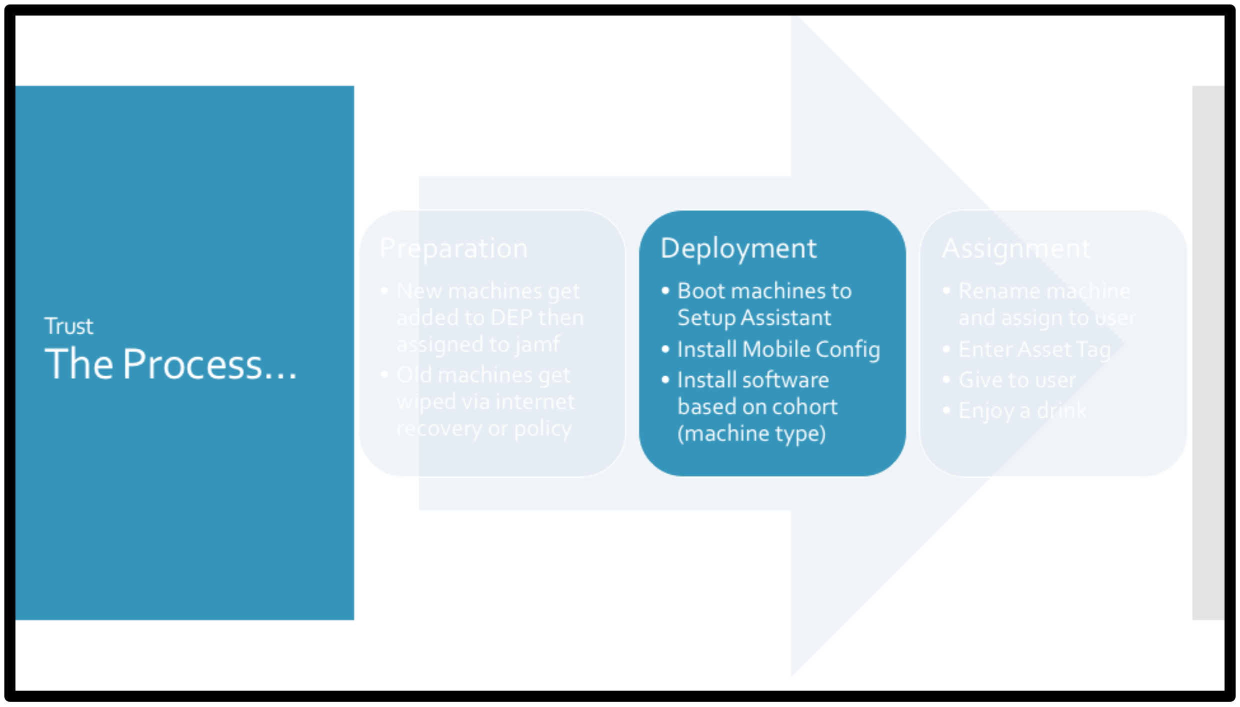
Deployment

- Boot machines to Setup Assistant
- Install Mobile Config
- Install software based on cohort (machine type)

Assignment

- Rename machine and assign to user
- Enter Asset Tag
- Give to user
- Enjoy a drink

So lets go back to our steps…we just prepped our machines and next we should deploy.



Trust

The Process…

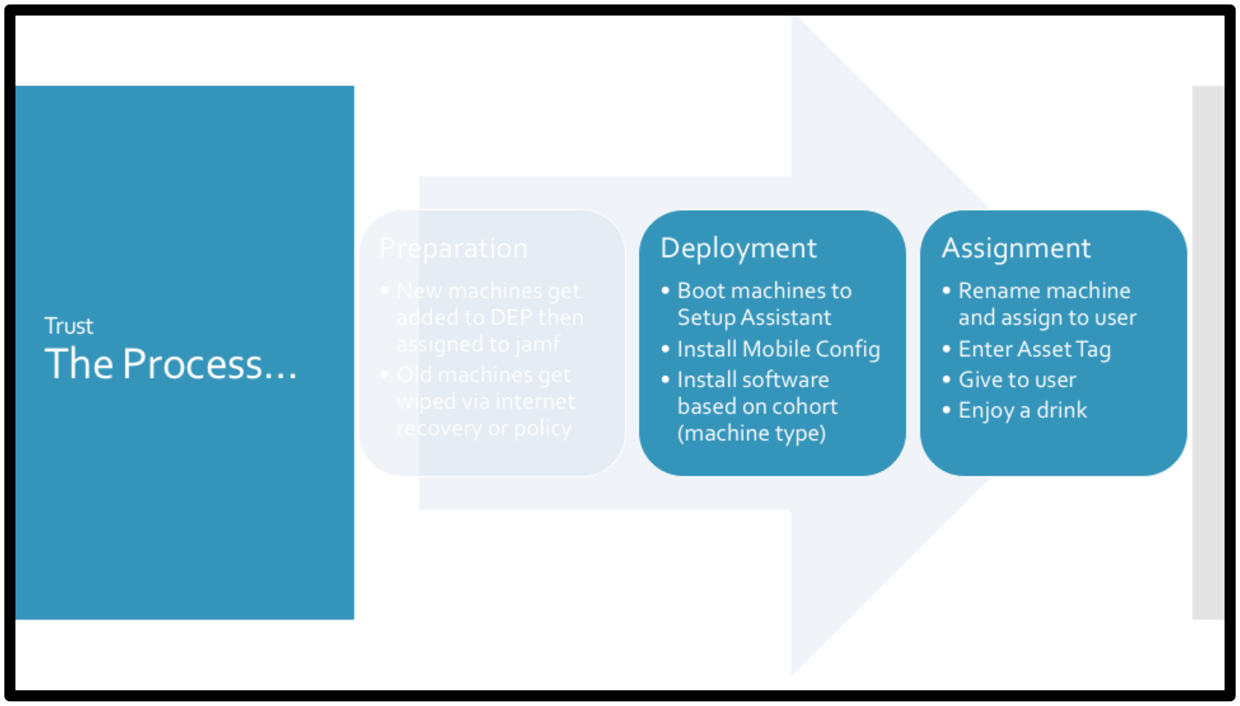
Preparation
- New machines get added to DEP then assigned to jamf
- Old machines get wiped via internet recovery or policy

Deployment
- Boot machines to Setup Assistant
- Install Mobile Config
- Install software based on cohort (machine type)

Assignment
- Rename machine and assign to user
- Enter Asset Tag
- Give to user
- Enjoy a drink

So deployment is simply booting the machine to get the mobile config and install the software we need based on the machine type. But what if you don't know who is getting the machine? How are you going to figure out what software gets installed?

We don't assign the machine until the next step..
_C_



Trust

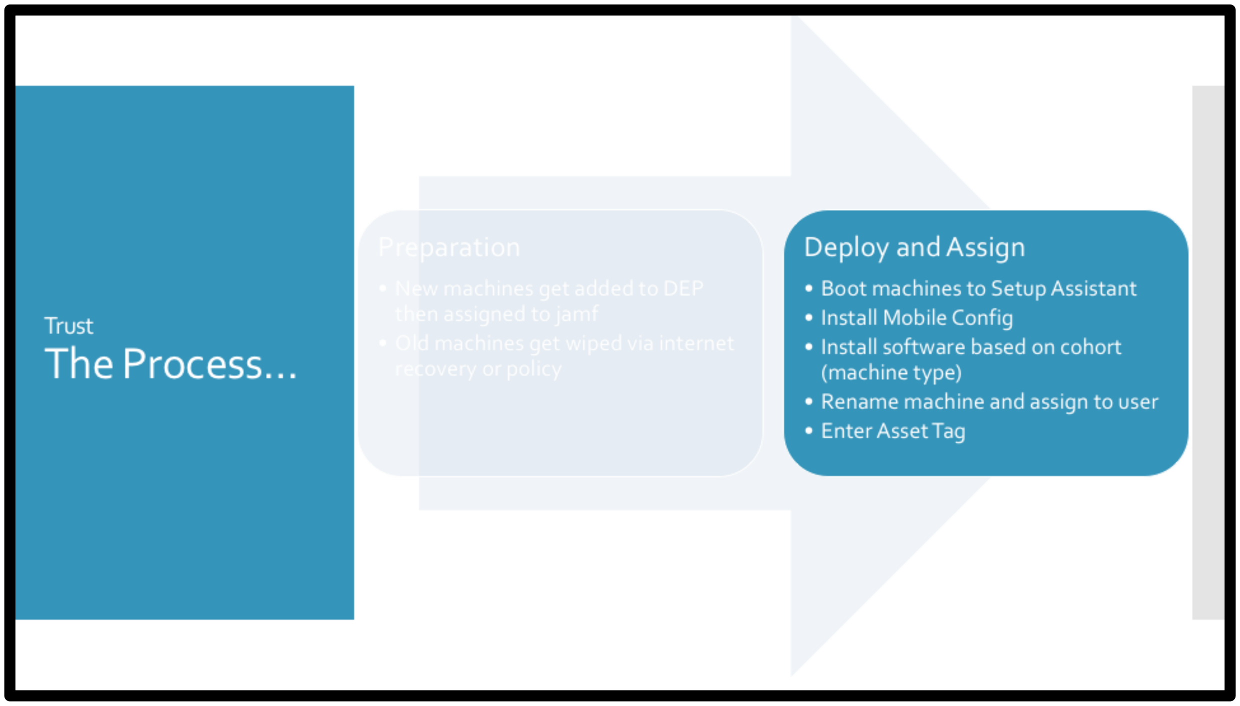
The Process…

Preparation
- New machines get added to DEP then assigned to jamf
- Old machines get wiped via internet recovery or policy

Deployment
- Boot machines to Setup Assistant
- Install Mobile Config
- Install software based on cohort (machine type)

Assignment
- Rename machine and assign to user
- Enter Asset Tag
- Give to user
- Enjoy a drink

Well, maybe we can do both at the same time… _C_



Trust

The Process…

Preparation

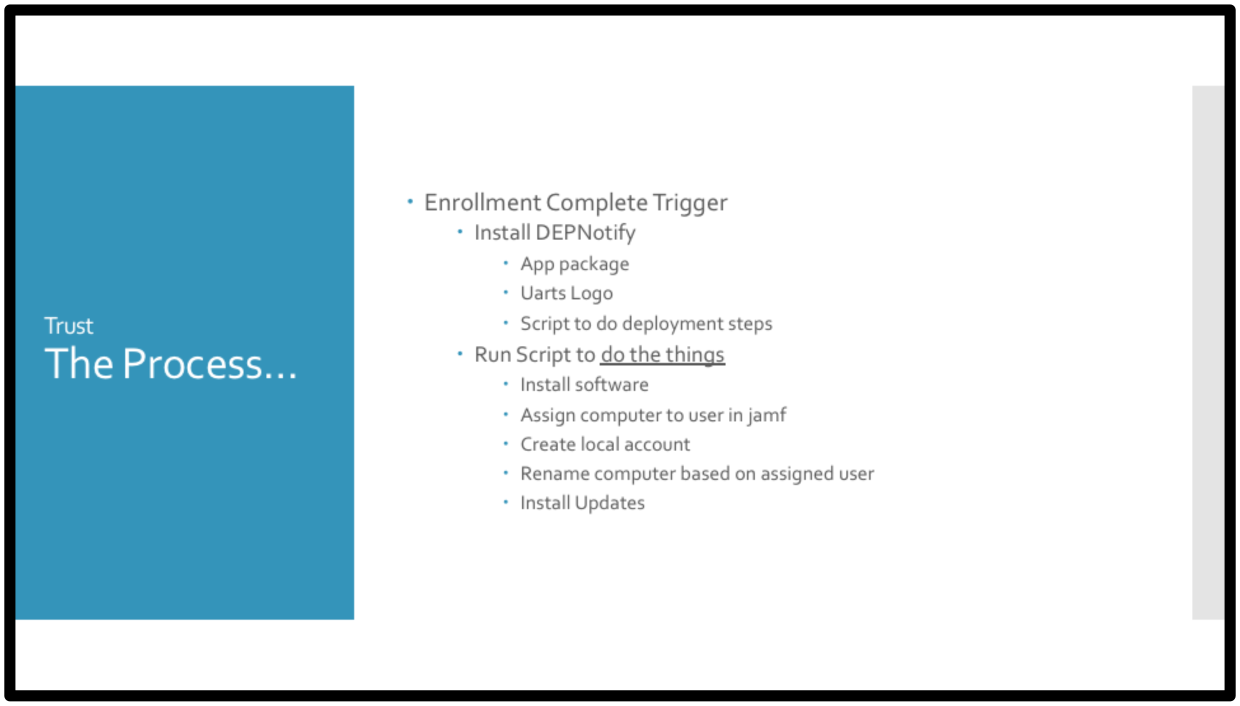
- New machines get added to DEP then assigned to jamf
- Old machines get wiped via internet recovery or policy

Deploy and Assign

- Boot machines to Setup Assistant
- Install Mobile Config
- Install software based on cohort (machine type)
- Rename machine and assign to user
- Enter Asset Tag

So, after we prepare a machine, that machine can either go into storage until it’s ready to be deployed (which happens often for us) or it can be deployed and assigned right away.

This will also lower the amount of steps needed to deploy the machine, then assign it, then rename the machine and add the user account, etc. We can take care of everything at one time.


Trust

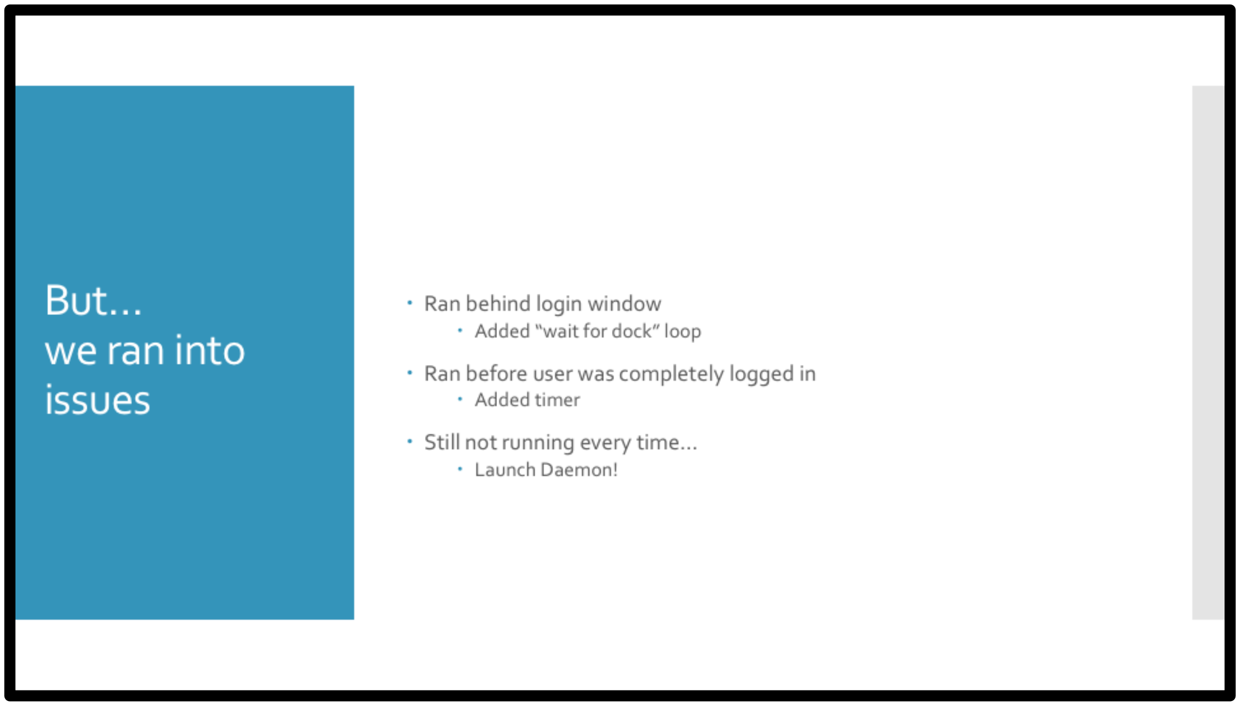
The Process…

- Enrollment Complete Trigger
  - Install DEPNotify
    - App package
    - Uarts Logo
    - Script to do deployment steps
  - Run Script to do the things
    - Install software
    - Assign computer to user in jamf
    - Create local account
    - Rename computer based on assigned user
    - Install Updates

So here is our enrollment complete process!

Install DEPNotify with extra payload items then run a script.

Sounds great, right?



But… we ran into issues

- Ran behind login window
  - Added "wait for dock" loop
- Ran before user was completely logged in
  - Added timer
- Still not running every time…
  - Launch Daemon!

It wasn't…we ran into issues.

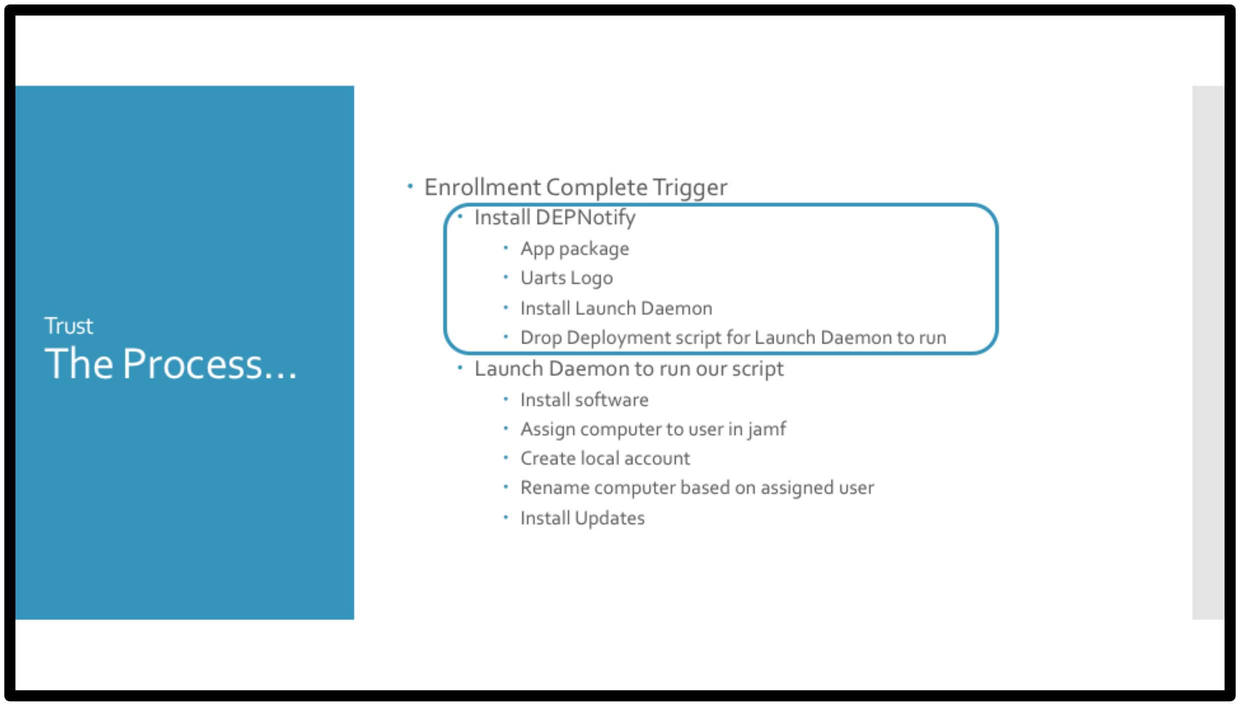
_C_ First we found it was running behind the login window. Things would work, but nothing would show up which was obviously bad.

_C_ So we added a "wait for dock loop". This worked about 60% of the time because the next issue was

_C_ It started running before the user was completely logged in. _C_ So we added a timer.

_C_ This too didn't run every time, we would have edge cases where it would again run but it be in the background or what if we had a tech close the laptop..then you would have the script waiting to run in the background.

So we decided to solve that with _C_ a launch daemon!


Trust

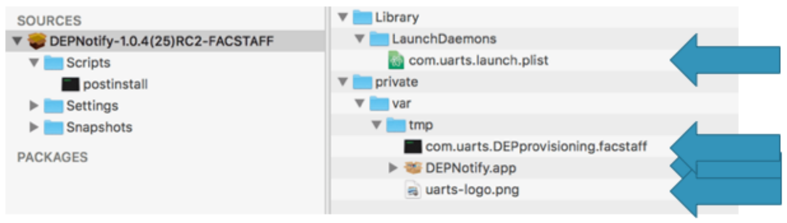
The Process…

- Enrollment Complete Trigger
  - Install DEPNotify
    - App package
    - Uarts Logo
    - Install Launch Daemon
    - Drop Deployment script for Launch Daemon to run
  - Launch Daemon to run our script
    - Install software
    - Assign computer to user in jamf
    - Create local account
    - Rename computer based on assigned user
    - Install Updates

So let’s go back and look at _C_ this step.

We’re still installing DEP with a payload but we’re going to add the deployment script and launch daemon.

Create DEPNotify Package

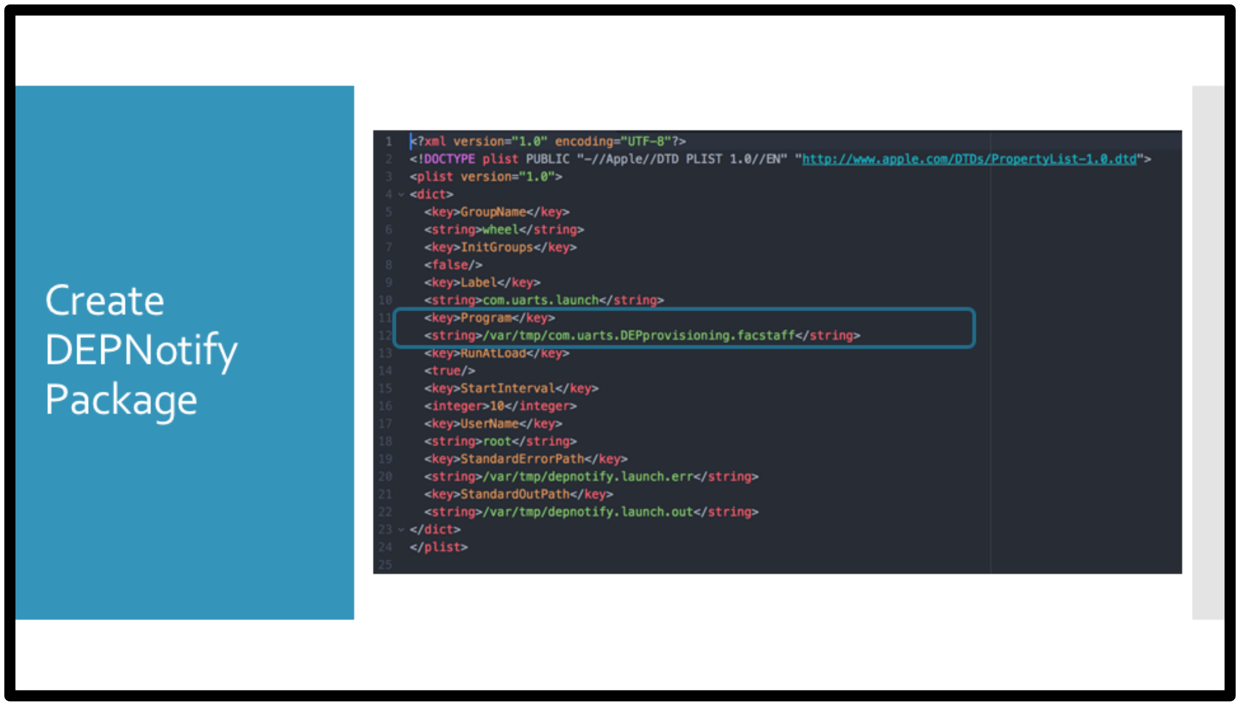


So here is our package.

_C_ Here is the app in /var/tmp

_C_ Our logo
_C_ And our provisioning script

That script gets called by _C_ our launch daemon.

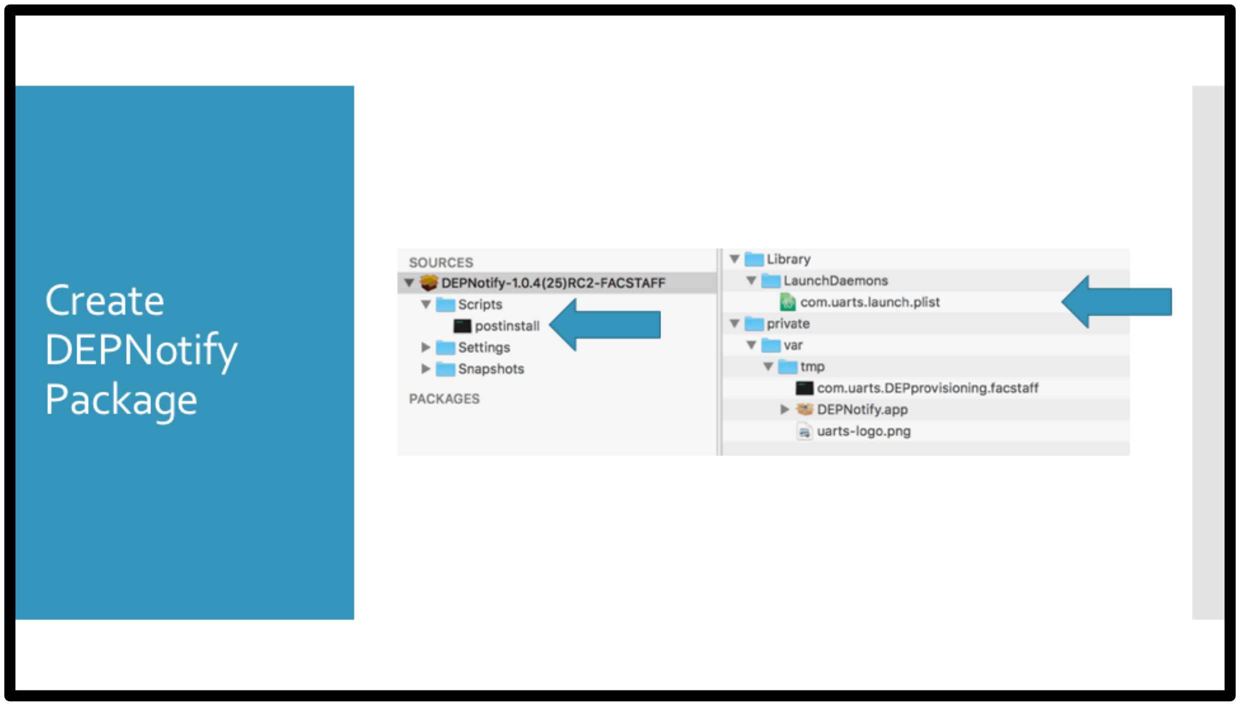


Create DEPNotify Package

```
<?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>GroupName</key>
  <string>wheel</string>
  <key>InitGroups</key>
  <false/>
  <key>Label</key>
  <string>com.uarts.launch</string>
  <key>Program</key>
  <string>/var/tmp/com.uarts.DEPprovisioning.facstaff</string>
  <key>RunAtLoad</key>
  <true/>
  <key>StartInterval</key>
  <integer>10</integer>
  <key>UserName</key>
  <string>root</string>
  <key>StandardErrorPath</key>
  <string>/var/tmp/depnotify.launch.err</string>
  <key>StandardOutPath</key>
  <string>/var/tmp/depnotify.launch.out</string>
</dict>
</plist>
```

This is our launch daemon here. Pretty straight forward, it loads _C_ the provisioning script 10 seconds after login is complete.

It will also run if the enrollment trigger occurs while the user is logged in, we have this issue quite often.



Create DEPNotify Package

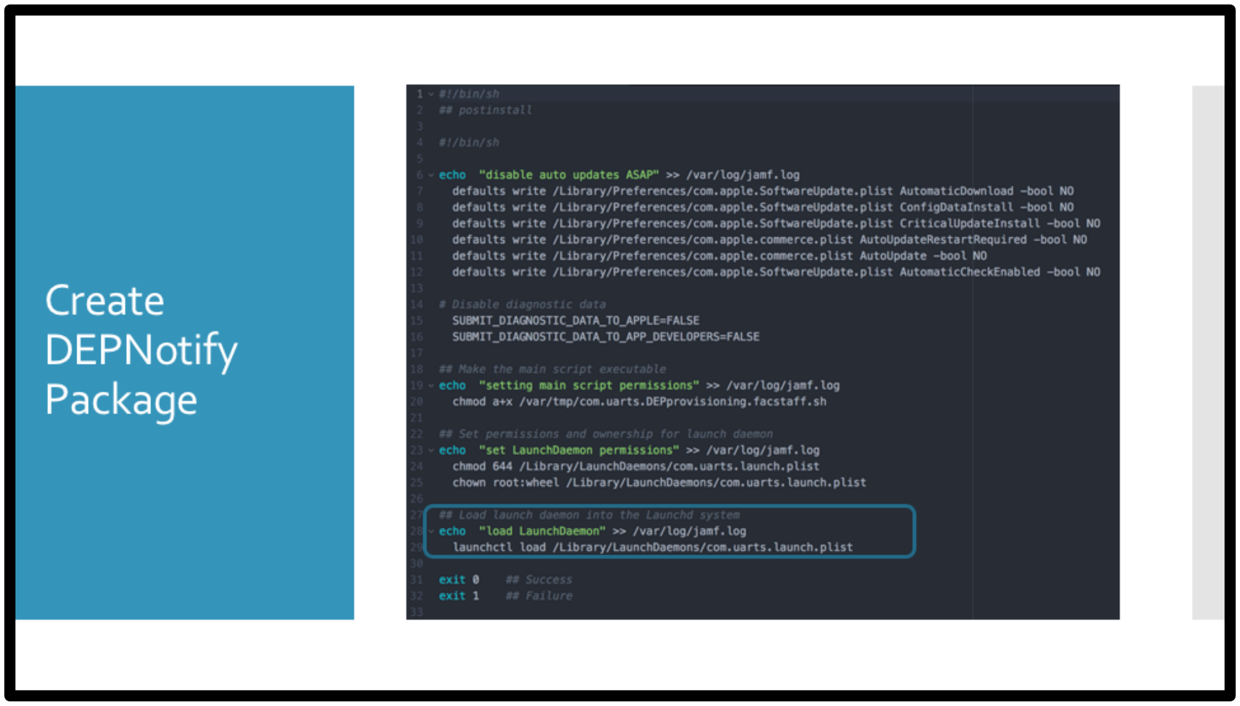
SOURCES
- DEPNotify-1.0.4(25)RC2-FACSTAFF
  - Scripts
    - postinstall
  - Settings
  - Snapshots

PACKAGES

```
Library
  LaunchDaemons
    com.uarts.launch.plist
private
  var
    tmp
      com.uarts.DEPprovisioning.facstaff
      DEPNotify.app
      uarts-logo.png
```

Now that we have that there, we have to load it and that is what

_C_ our post install script is for.

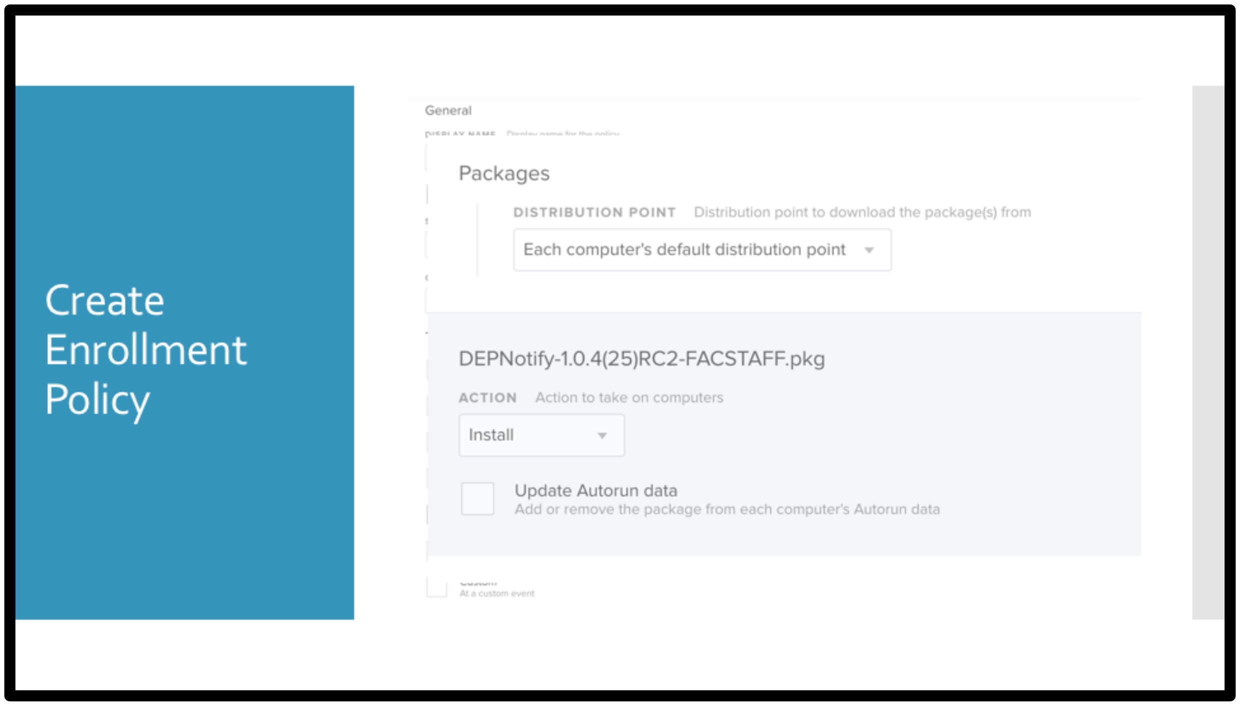


Create DEPNotify Package

```
#!/bin/sh
## postinstall

#!/bin/sh

echo  "disable auto updates ASAP" >> /var/log/jamf.log
  defaults write /Library/Preferences/com.apple.SoftwareUpdate.plist AutomaticDownload -bool NO
  defaults write /Library/Preferences/com.apple.SoftwareUpdate.plist ConfigDataInstall -bool NO
  defaults write /Library/Preferences/com.apple.SoftwareUpdate.plist CriticalUpdateInstall -bool NO
  defaults write /Library/Preferences/com.apple.commerce.plist AutoUpdateRestartRequired -bool NO
  defaults write /Library/Preferences/com.apple.commerce.plist AutoUpdate -bool NO
  defaults write /Library/Preferences/com.apple.SoftwareUpdate.plist AutomaticCheckEnabled -bool NO

# Disable diagnostic data
  SUBMIT_DIAGNOSTIC_DATA_TO_APPLE=FALSE
  SUBMIT_DIAGNOSTIC_DATA_TO_APP_DEVELOPERS=FALSE

## Make the main script executable
echo  "setting main script permissions" >> /var/log/jamf.log
  chmod a+x /var/tmp/com.uarts.DEPprovisioning.facstaff.sh

## Set permissions and ownership for launch daemon
echo  "set LaunchDaemon permissions" >> /var/log/jamf.log
  chmod 644 /Library/LaunchDaemons/com.uarts.launch.plist
  chown root:wheel /Library/LaunchDaemons/com.uarts.launch.plist

## Load launch daemon into the Launchd system
echo  "load LaunchDaemon" >> /var/log/jamf.log
  launchctl load /Library/LaunchDaemons/com.uarts.launch.plist

exit 0    ## Success
exit 1    ## Failure

```

Again, pretty straight forward. Most of this is just extra stuff to disable any auto updates, the most important line is here _C_ just loading our launch daemon.

Now our package is done..



Create Enrollment Policy

General

Packages

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

Each computer's default distribution point

DEPNotify-1.0.4(25)RC2-FACSTAFF.pkg

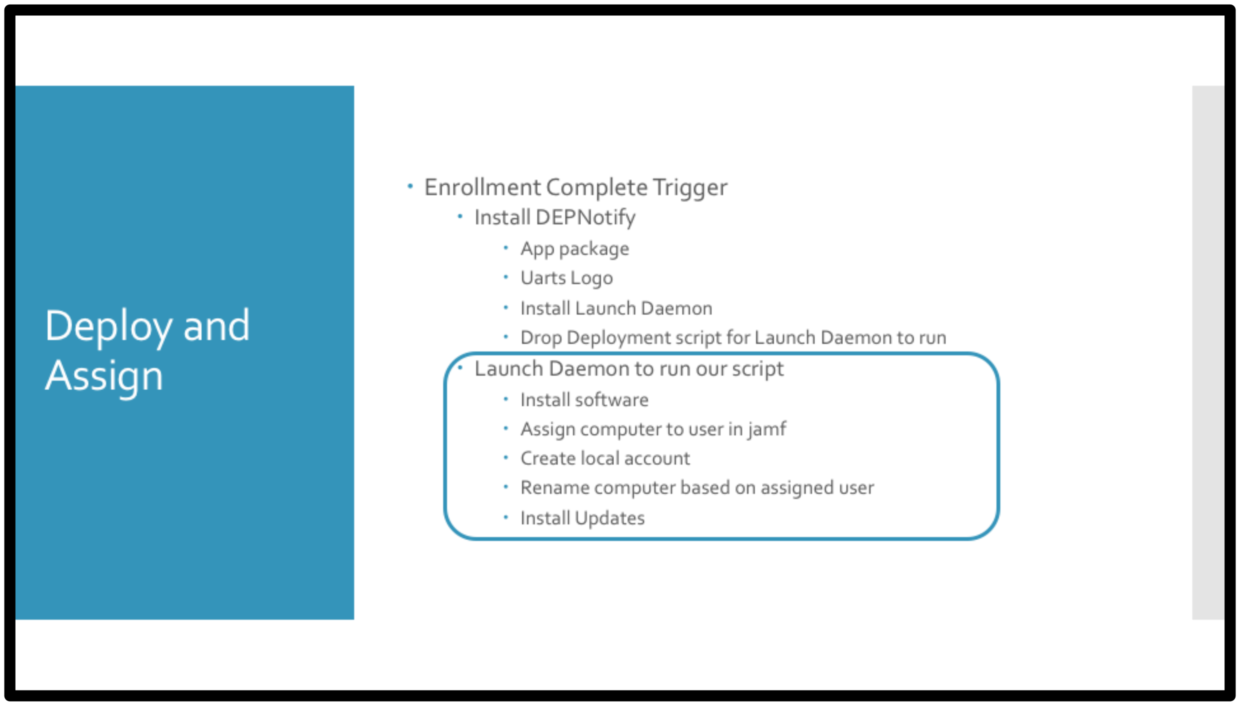
ACTION Action to take on computers

Install

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

At a custom event

Create our enrollment policy. Trigger is enrollment complete and we're just installing our custom PKG. _C_



Deploy and Assign

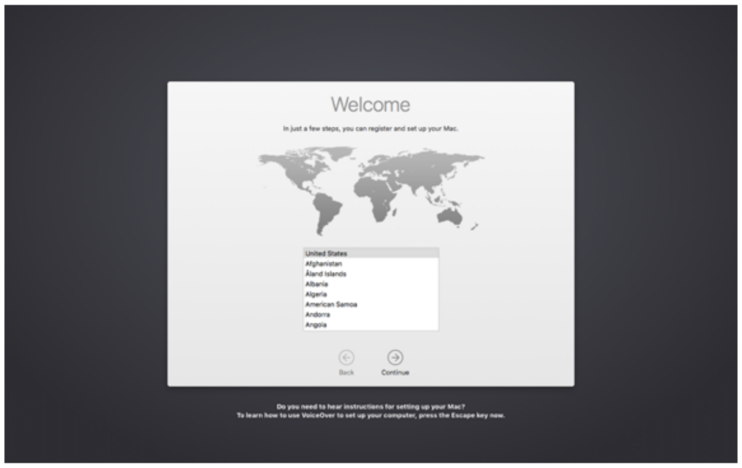
- Enrollment Complete Trigger
  - Install DEPNotify
    - App package
    - Uarts Logo
    - Install Launch Daemon
    - Drop Deployment script for Launch Daemon to run
  - Launch Daemon to run our script
    - Install software
    - Assign computer to user in jamf
    - Create local account
    - Rename computer based on assigned user
    - Install Updates

So once DEPNotify is installed the only thing left to do is _C_

Have our launch daemon run our provisioning script.

Let’s see that in action.

Putting it all together

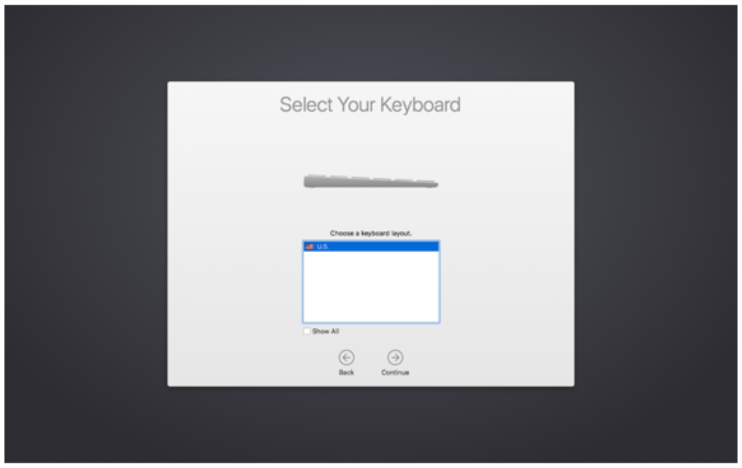

Welcome

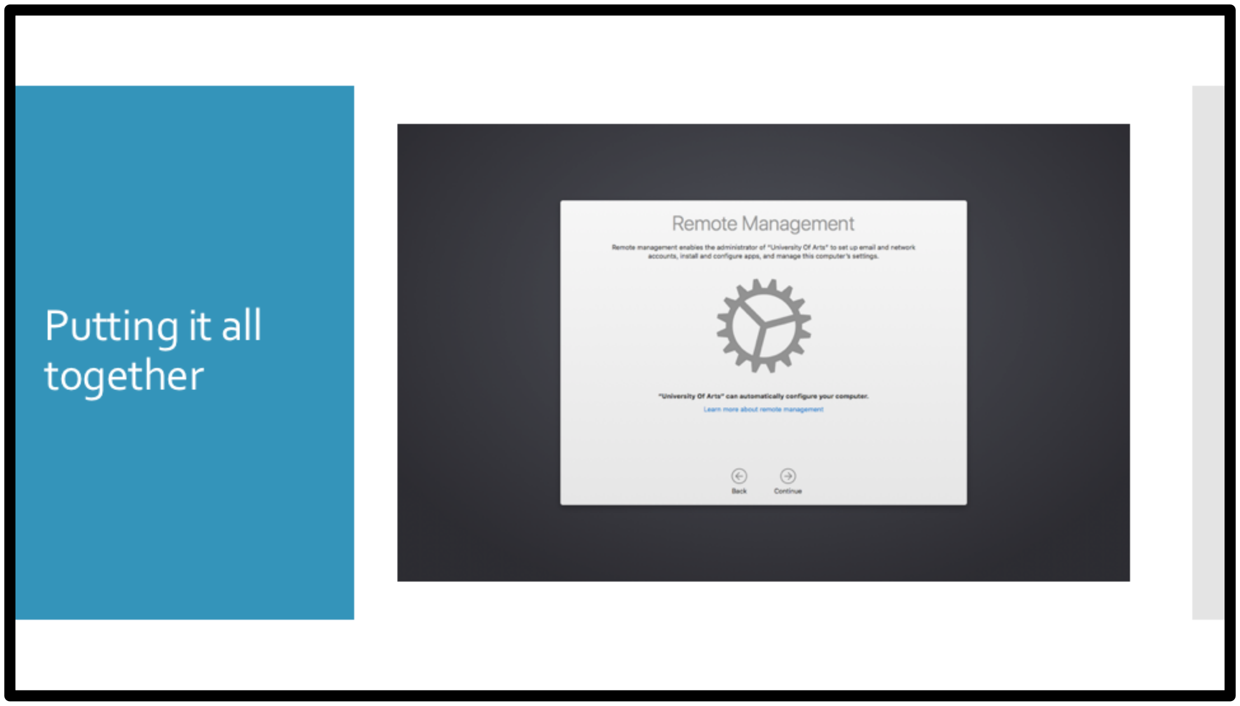
In just a few steps, you can register and set up your Mac.

Do you need to hear instructions for setting up your Mac?
To learn how to use VoiceOver to set up your computer, press the Escape key now.

Okay, after we do our first boot we go through the setup assistant.

Putting it all together





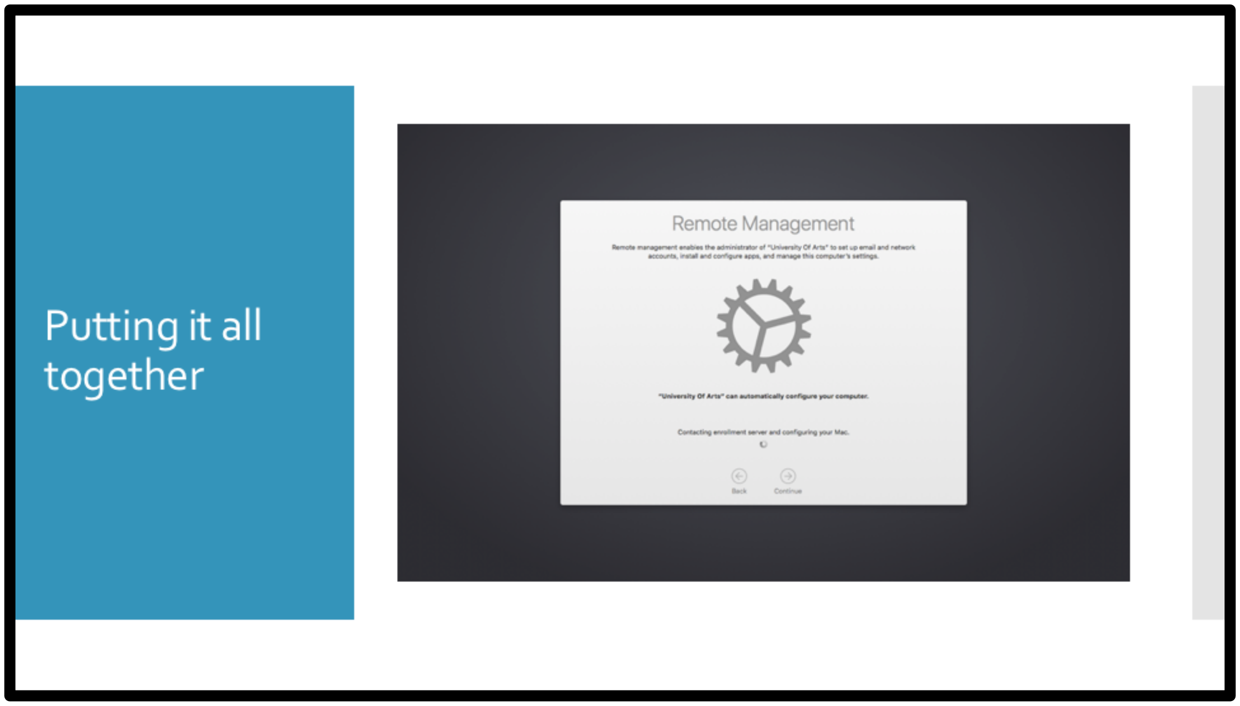
Putting it all together

Remote Management

Remote management enables the administrator of "University Of Arts" to set up email and network accounts, install and configure apps, and manage this computer's settings.

"University Of Arts" can automatically configure your computer.

Now, we know this screen, it means we've got a DEP machine that's tied to a server and we're ready to roll! Clicking continue…



Putting it all together

Remote Management

Remote management enables the administrator of "University Of Arts" to set up email and network accounts, install and configure apps, and manage this computer's settings.

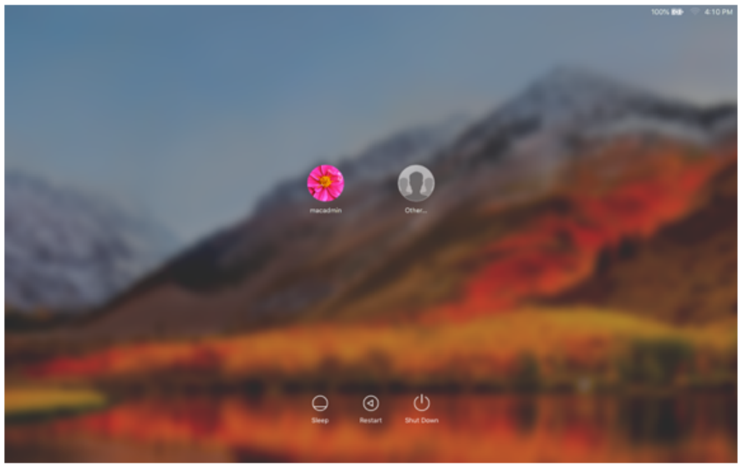
"University Of Arts" can automatically configure your computer.

Contacting enrollment server and configuring your Mac.

Not sure if anyone else has the same issue we do but sometimes this screen can take up to 3 minutes… it's weird…but it eventually completes

When it's done doing this we're greeted with our login window.

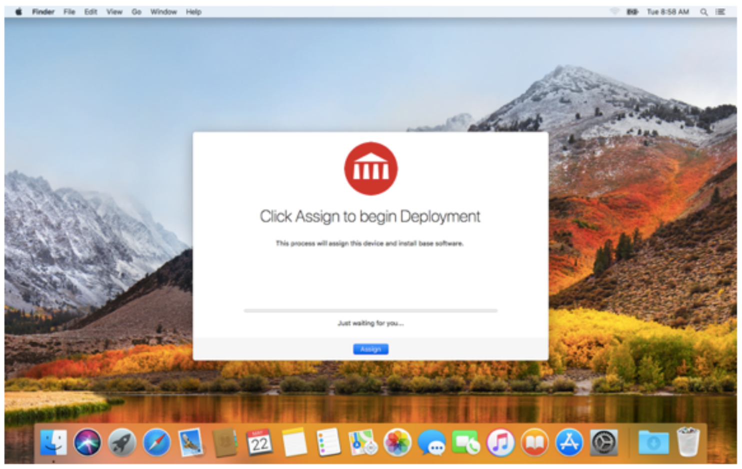
Putting it all together



Notice our macadmin account is there and it didn't ask for any authentication.

So we'll log in.

Putting it all together

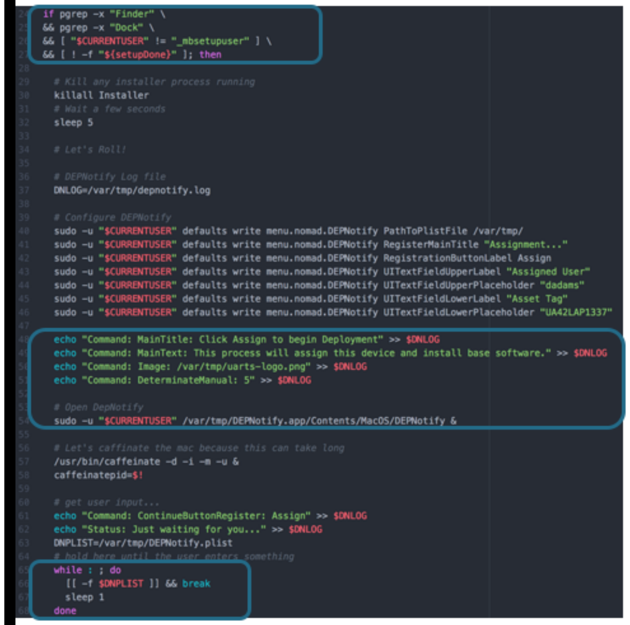


Click Assign to begin Deployment

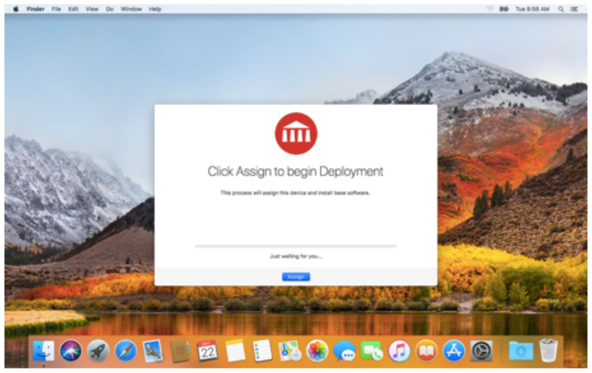
This process will assign this device and install base software.

Just waiting for you...

And this is what we’re greeted to after login (or after the enrollment policy runs AFTER logging in), our DEPNotify window. Let’s see it with our code side-by side.



```
if pgrep -x "Finder" \
&& pgrep -x "Dock" \
&& [ "$CURRENTUSER" != "_mbsetupuser" ] \
&& [ ! -f "${setupDone}" ]; then

  # Kill any installer process running
  killall Installer
  # Wait a few seconds
  sleep 5

  # Let's Roll!

  # DEPNotify Log file
  DNLOG=/var/tmp/depnotify.log

  # Configure DEPNotify
  sudo -u "$CURRENTUSER" defaults write menu.nomad.DEPNotify PathToPlistFile /var/tmp/
  sudo -u "$CURRENTUSER" defaults write menu.nomad.DEPNotify RegisterMainTitle "Assignment..."
  sudo -u "$CURRENTUSER" defaults write menu.nomad.DEPNotify RegistrationButtonLabel Assign
  sudo -u "$CURRENTUSER" defaults write menu.nomad.DEPNotify UITextFieldUpperLabel "Assigned User"
  sudo -u "$CURRENTUSER" defaults write menu.nomad.DEPNotify UITextFieldUpperPlaceholder "dadams"
  sudo -u "$CURRENTUSER" defaults write menu.nomad.DEPNotify UITextFieldLowerLabel "Asset Tag"
  sudo -u "$CURRENTUSER" defaults write menu.nomad.DEPNotify UITextFieldLowerPlaceholder "UA42LAP1337"

  echo "Command: MainTitle: Click Assign to begin Deployment" >> $DNLOG
  echo "Command: MainText: This process will assign this device and install base software." >> $DNLOG
  echo "Command: Image: /var/tmp/uarts-logo.png" >> $DNLOG
  echo "Command: DeterminateManual: 5" >> $DNLOG

  # Open DepNotify
  sudo -u "$CURRENTUSER" /var/tmp/DEPNotify.app/Contents/MacOS/DEPNotify &

  # Let's caffinate the mac because this can take long
  /usr/bin/caffeinate -d -i -m -u &
  caffeinatepid=$!

  # get user input...
  echo "Command: ContinueButtonRegister: Assign" >> $DNLOG
  echo "Status: Just waiting for you..." >> $DNLOG
  DNPLIST=/var/tmp/DEPNotify.plist
  # hold here until the user enters something
  while : ; do
    [[ -f $DNPLIST ]] && break
    sleep 1
  done
```



Click Assign to begin Deployment

This process will assign this device and install base software.

Just waiting for you...

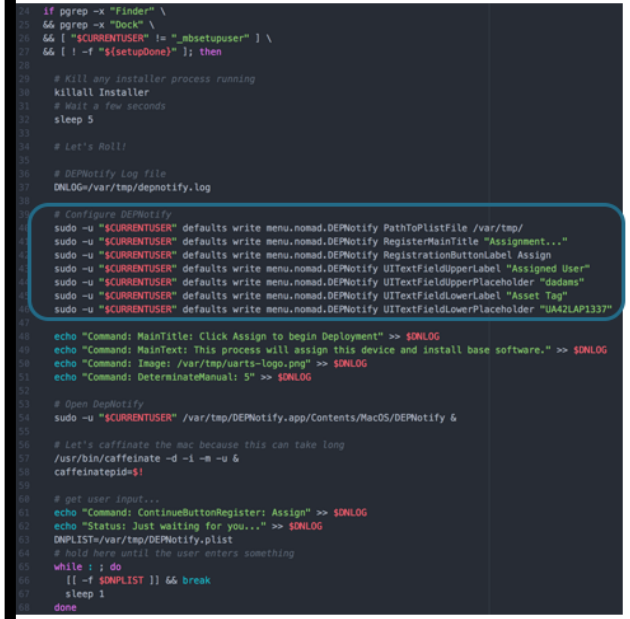
Here's our code. Now, DEP notify was only launched because _C_

we waited for the dock and finder to be loaded and made sure we're not in the mbsetupuser account… we also are checking for this setup done file, more on that later.

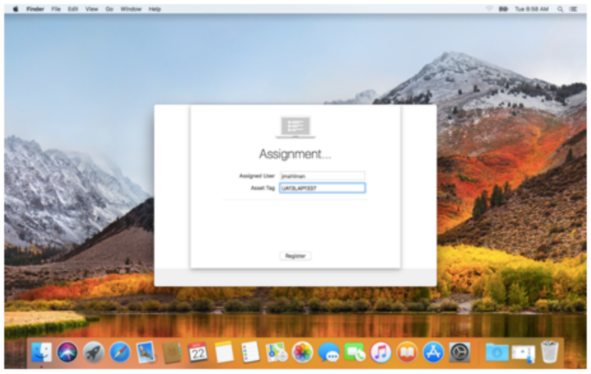
_C_ The window looks the way it does because we sent these commands to the depnotify log file. Then we launched it AFTER sending that to the log.

Now the interesting thig is that this is now just waiting for the user to hit the "Assign" button. I set it up this way so our tech's don't have to wait around until it's done installing things, nothing will happen until they start. So that's down here _C_.

It's just waiting for the "DNPLIST" which is a file that gets dropped after input is complete. Which is next up…

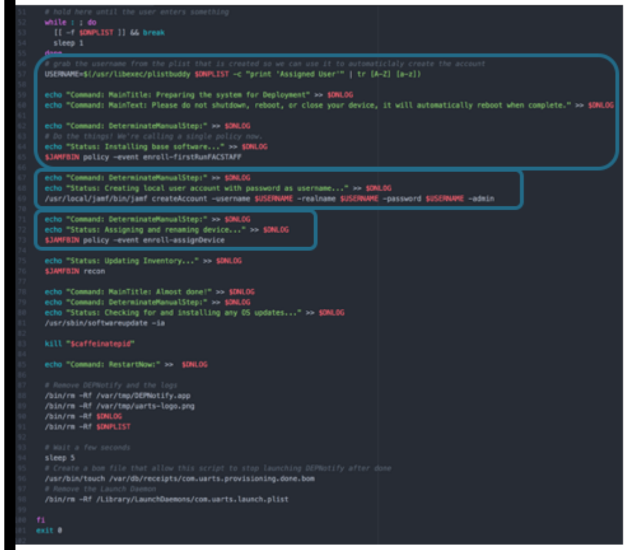


```
if pgrep -x "Finder" \
&& pgrep -x "Dock" \
&& [ "$CURRENTUSER" != "_mbsetupuser" ] \
&& [ ! -f "${setupDone}" ]; then

  # Kill any installer process running
  killall Installer
  # Wait a few seconds
  sleep 5

  # Let's Roll!

  # DEPNotify Log file
  DNLOG=/var/tmp/depnotify.log

  # Configure DEPNotify
  sudo -u "$CURRENTUSER" defaults write menu.nomad.DEPNotify PathToPlistFile /var/tmp/
  sudo -u "$CURRENTUSER" defaults write menu.nomad.DEPNotify RegisterMainTitle "Assignment..."
  sudo -u "$CURRENTUSER" defaults write menu.nomad.DEPNotify RegistrationButtonLabel Assign
  sudo -u "$CURRENTUSER" defaults write menu.nomad.DEPNotify UITextFieldUpperLabel "Assigned User"
  sudo -u "$CURRENTUSER" defaults write menu.nomad.DEPNotify UITextFieldUpperPlaceholder "dadams"
  sudo -u "$CURRENTUSER" defaults write menu.nomad.DEPNotify UITextFieldLowerLabel "Asset Tag"
  sudo -u "$CURRENTUSER" defaults write menu.nomad.DEPNotify UITextFieldLowerPlaceholder "UA42LAP1337"

  echo "Command: MainTitle: Click Assign to begin Deployment" >> $DNLOG
  echo "Command: MainText: This process will assign this device and install base software." >> $DNLOG
  echo "Command: Image: /var/tmp/uarts-logo.png" >> $DNLOG
  echo "Command: DeterminateManual: 5" >> $DNLOG

  # Open DepNotify
  sudo -u "$CURRENTUSER" /var/tmp/DEPNotify.app/Contents/MacOS/DEPNotify &

  # Let's caffinate the mac because this can take long
  /usr/bin/caffeinate -d -i -m -u &
  caffeinatepid=$!

  # get user input...
  echo "Command: ContinueButtonRegister: Assign" >> $DNLOG
  echo "Status: Just waiting for you..." >> $DNLOG
  DNPLIST=/var/tmp/DEPNotify.plist
  # hold here until the user enters something
  while : ; do
    [[ -f $DNPLIST ]] && break
    sleep 1
  done
```

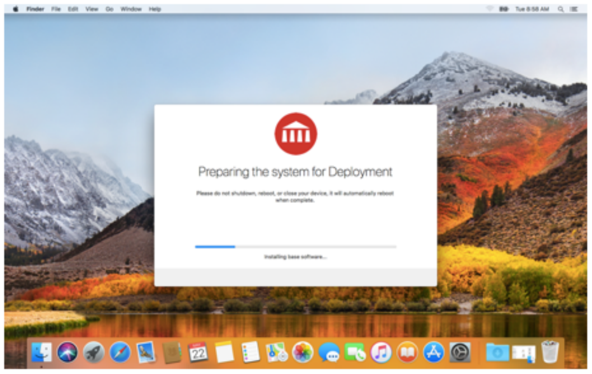


When the tech clicks "Assign" the registration window drops down.

That is populated by this section of code. Notice that the code was run before DEPNotify was open because it's actually just setting some prefs up. We only need two text fields, but you can also add a dropdown menu or two and a "sensitive data" checkbox. New features are being added every week it seems.



```
# hold here until the user enters something
while : ; do
  [[ -f $DNPLIST ]] && break
  sleep 1
done
# grab the username from the plist that is created so we can use it to automaticlaly create the account
USERNAME=$(/usr/libexec/plistbuddy $DNPLIST -c "print 'Assigned User'" | tr [A-Z] [a-z])

echo "Command: MainTitle: Preparing the system for Deployment" >> $DNLOG
echo "Command: MainText: Please do not shutdown, reboot, or close your device, it will automatically reboot when complete." >> $DNLOG

echo "Command: DeterminateManualStep:" >> $DNLOG
# Do the things! We're calling a single policy now.
echo "Status: Installing base software..." >> $DNLOG
$JAMFBIN policy -event enroll-firstRunFACSTAFF

echo "Command: DeterminateManualStep:" >> $DNLOG
echo "Status: Creating local user account with password as username..." >> $DNLOG
/usr/local/jamf/bin/jamf createAccount -username $USERNAME -realname $USERNAME -password $USERNAME -admin

echo "Command: DeterminateManualStep:" >> $DNLOG
echo "Status: Assigning and renaming device..." >> $DNLOG
$JAMFBIN policy -event enroll-assignDevice

echo "Status: Updating Inventory..." >> $DNLOG
$JAMFBIN recon

echo "Command: MainTitle: Almost done!" >> $DNLOG
echo "Command: DeterminateManualStep:" >> $DNLOG
echo "Status: Checking for and installing any OS updates..." >> $DNLOG
/usr/sbin/softwareupdate -ia

kill "$caffeinatepid"

echo "Command: RestartNow:" >>  $DNLOG

# Remove DEPNotify and the logs
/bin/rm -Rf /var/tmp/DEPNotify.app
/bin/rm -Rf /var/tmp/uarts-logo.png
/bin/rm -Rf $DNLOG
/bin/rm -Rf $DNPLIST

# Wait a few seconds
sleep 5
# Create a bom file that allow this script to stop launching DEPNotify after done
/usr/bin/touch /var/db/receipts/com.uarts.provisioning.done.bom
# Remove the Launch Daemon
/bin/rm -Rf /Library/LaunchDaemons/com.uarts.launch.plist

fi
exit 0
```



Preparing the system for Deployment

Please do not shutdown, reboot, or close your device, it will automatically reboot when complete.

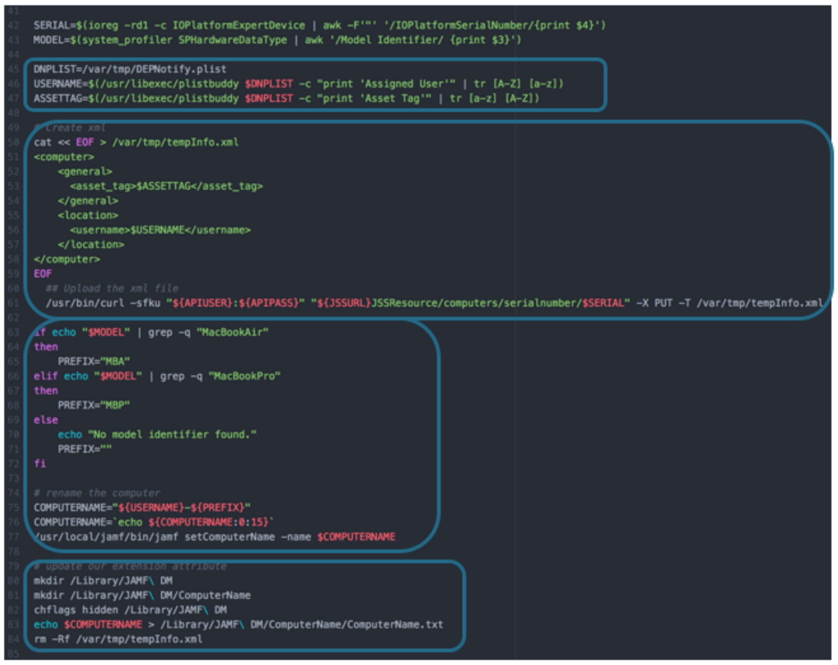
Installing base software...

So after the user input is done we can begin the tasks.

_C_ First we're grabbing the username from the plist that was created after registration and holding it for creating the local account later. Then we're passing more commands into the DEPNotify log file, so now our window has changed. And then we run our first run policy.

_C_ After the policy is run we are creating the local account based on the assigned user and setting the password to the username (temporarily).

_C_ Then we're going to assign and rename the device with another policy. This policy runs a script…



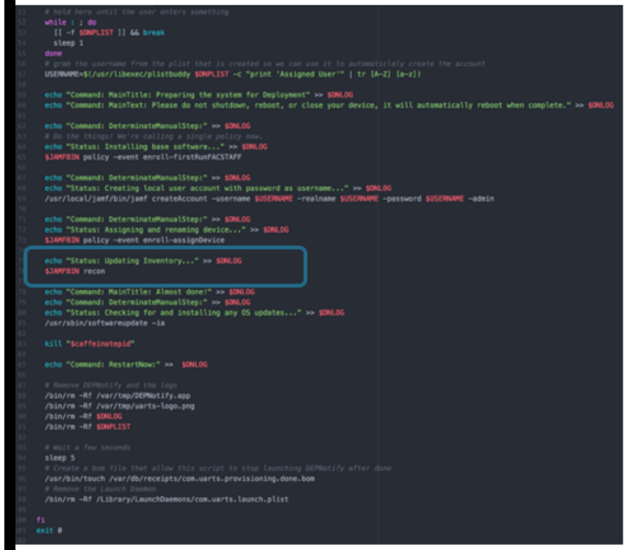
```
SERIAL=$(ioreg -rd1 -c IOPlatformExpertDevice | awk -F'"' '/IOPlatformSerialNumber/{print $4}')
MODEL=$(system_profiler SPHardwareDataType | awk '/Model Identifier/ {print $3}')

DNPLIST=/var/tmp/DEPNotify.plist
USERNAME=$(/usr/libexec/plistbuddy $DNPLIST -c "print 'Assigned User'" | tr [A-Z] [a-z])
ASSETTAG=$(/usr/libexec/plistbuddy $DNPLIST -c "print 'Asset Tag'" | tr [a-z] [A-Z])

# Create xml
cat << EOF > /var/tmp/tempInfo.xml
<computer>
    <general>
      <asset_tag>$ASSETTAG</asset_tag>
    </general>
    <location>
      <username>$USERNAME</username>
    </location>
</computer>
EOF
  ## Upload the xml file
  /usr/bin/curl -sfku "${APIUSER}:${APIPASS}" "${JSSURL}JSSResource/computers/serialnumber/$SERIAL" -X PUT -T /var/tmp/tempInfo.xml

if echo "$MODEL" | grep -q "MacBookAir"
then
    PREFIX="MBA"
elif echo "$MODEL" | grep -q "MacBookPro"
then
    PREFIX="MBP"
else
    echo "No model identifier found."
    PREFIX=""
fi

# rename the computer
COMPUTERNAME="${USERNAME}-${PREFIX}"
COMPUTERNAME=`echo ${COMPUTERNAME:0:15}`
/usr/local/jamf/bin/jamf setComputerName -name $COMPUTERNAME

# update our extension attribute
mkdir /Library/JAMF\ DM
mkdir /Library/JAMF\ DM/ComputerName
chflags hidden /Library/JAMF\ DM
echo $COMPUTERNAME > /Library/JAMF\ DM/ComputerName/ComputerName.txt
rm -Rf /var/tmp/tempInfo.xml
```

It just reads _C_ the plist file that was dropped at registration and sets variables for user and asset tag.

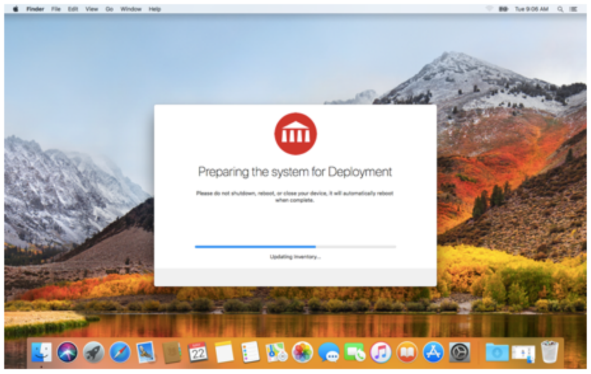
_C_ sets up an xml file and passes it to the jamf API

_C_ then it renames the machine using the assigned username with a prefix for MBA or MBP.

_C_ Finally, we update our extension attribute receipts on the machine and remove the temp XML file.



```
    # hold here until the user enters something
    while : ; do
      [[ -f $DNPLIST ]] && break
      sleep 1
    done
    # grab the username from the plist that is created so we can use it to automaticlaly create the account
    USERNAME=$(/usr/libexec/plistbuddy $DNPLIST -c "print 'Assigned User'" | tr [A-Z] [a-z])

    echo "Command: MainTitle: Preparing the system for Deployment" >> $DNLOG
    echo "Command: MainText: Please do not shutdown, reboot, or close your device, it will automatically reboot when complete." >> $DNLOG

    echo "Command: DeterminateManualStep:" >> $DNLOG
    # Do the things! We're calling a single policy now.
    echo "Status: Installing base software..." >> $DNLOG
    $JAMFBIN policy -event enroll-firstRunFACSTAFF

    echo "Command: DeterminateManualStep:" >> $DNLOG
    echo "Status: Creating local user account with password as username..." >> $DNLOG
    /usr/local/jamf/bin/jamf createAccount -username $USERNAME -realname $USERNAME -password $USERNAME -admin

    echo "Command: DeterminateManualStep:" >> $DNLOG
    echo "Status: Assigning and renaming device..." >> $DNLOG
    $JAMFBIN policy -event enroll-assignDevice

    echo "Status: Updating Inventory..." >> $DNLOG
    $JAMFBIN recon

    echo "Command: MainTitle: Almost done!" >> $DNLOG
    echo "Command: DeterminateManualStep:" >> $DNLOG
    echo "Status: Checking for and installing any OS updates..." >> $DNLOG
    /usr/sbin/softwareupdate -ia

    kill "$caffeinatepid"

    echo "Command: RestartNow:" >>  $DNLOG

    # Remove DEPNotify and the logs
    /bin/rm -Rf /var/tmp/DEPNotify.app
    /bin/rm -Rf /var/tmp/uarts-logo.png
    /bin/rm -Rf $DNLOG
    /bin/rm -Rf $DNPLIST

    # Wait a few seconds
    sleep 5
    # Create a bom file that allow this script to stop launching DEPNotify after done
    /usr/bin/touch /var/db/receipts/com.uarts.provisioning.done.bom
    # Remove the Launch Daemon
    /bin/rm -Rf /Library/LaunchDaemons/com.uarts.launch.plist

  fi
  exit 0
```

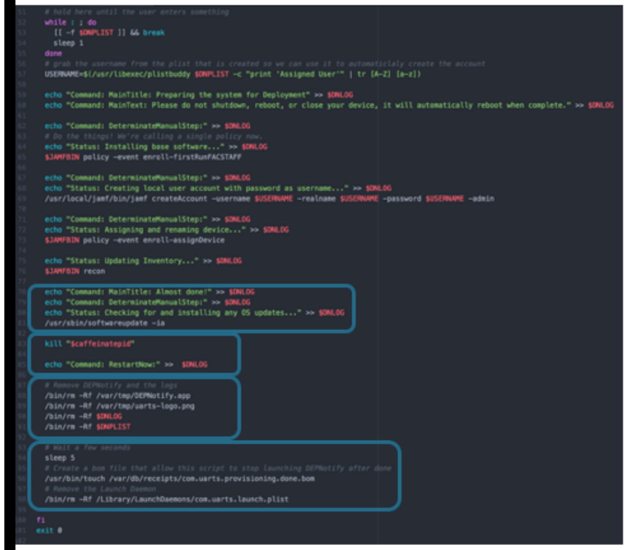


Preparing the system for Deployment

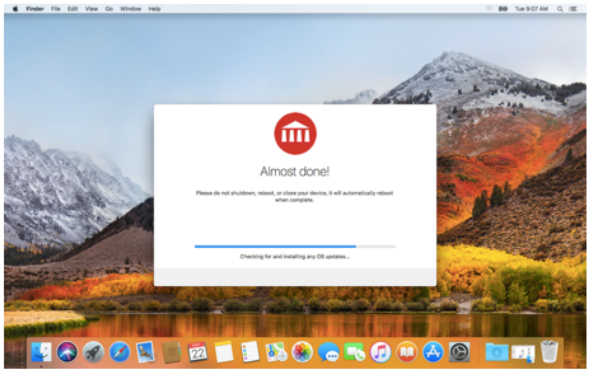
Please do not shutdown, reboot, or close your device, it will automatically reboot when complete.

Updating Inventory...

Back in this script we now run a recon to get all of that new info into our server



```
  # hold here until the user enters something
  while : ; do
    [[ -f $DNPLIST ]] && break
    sleep 1
  done
  # grab the username from the plist that is created so we can use it to automaticlaly create the account
  USERNAME=$(/usr/libexec/plistbuddy $DNPLIST -c "print 'Assigned User'" | tr [A-Z] [a-z])

  echo "Command: MainTitle: Preparing the system for Deployment" >> $DNLOG
  echo "Command: MainText: Please do not shutdown, reboot, or close your device, it will automatically reboot when complete." >> $DNLOG

  echo "Command: DeterminateManualStep:" >> $DNLOG
  # Do the things! We're calling a single policy now.
  echo "Status: Installing base software..." >> $DNLOG
  $JAMFBIN policy -event enroll-firstRunFACSTAFF

  echo "Command: DeterminateManualStep:" >> $DNLOG
  echo "Status: Creating local user account with password as username..." >> $DNLOG
  /usr/local/jamf/bin/jamf createAccount -username $USERNAME -realname $USERNAME -password $USERNAME -admin

  echo "Command: DeterminateManualStep:" >> $DNLOG
  echo "Status: Assigning and renaming device..." >> $DNLOG
  $JAMFBIN policy -event enroll-assignDevice

  echo "Status: Updating Inventory..." >> $DNLOG
  $JAMFBIN recon

  echo "Command: MainTitle: Almost done!" >> $DNLOG
  echo "Command: DeterminateManualStep:" >> $DNLOG
  echo "Status: Checking for and installing any OS updates..." >> $DNLOG
  /usr/sbin/softwareupdate -ia

  kill "$caffeinatepid"

  echo "Command: RestartNow:" >>  $DNLOG

  # Remove DEPNotify and the logs
  /bin/rm -Rf /var/tmp/DEPNotify.app
  /bin/rm -Rf /var/tmp/uarts-logo.png
  /bin/rm -Rf $DNLOG
  /bin/rm -Rf $DNPLIST

  # Wait a few seconds
  sleep 5
  # Create a bom file that allow this script to stop launching DEPNotify after done
  /usr/bin/touch /var/db/receipts/com.uarts.provisioning.done.bom
  # Remove the Launch Daemon
  /bin/rm -Rf /Library/LaunchDaemons/com.uarts.launch.plist

fi
exit 0
```



Almost done!

Please do not shutdown, reboot, or close your device, it will automatically reboot when complete.

Checking for and installing any OS updates...

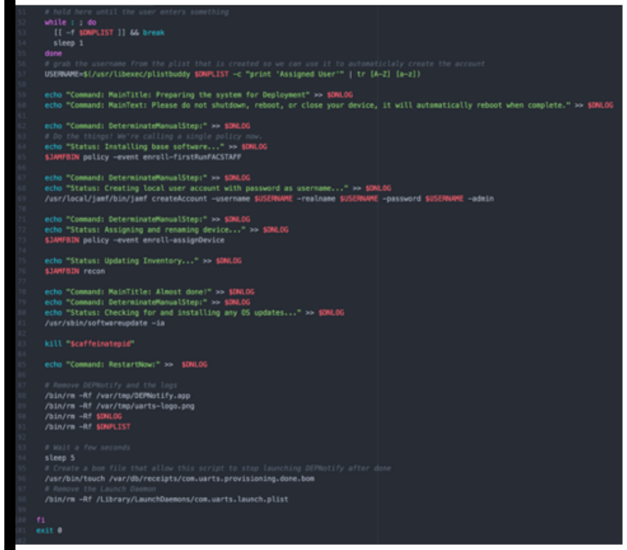
And next we run software updates..we’re also changing the text in our DEPNotify window just to tell the tech that it’s almost done.

_C_ After software updates install we send the command to DEPNotify to restart the machine right away, you can ask the user to hit "OK" to reboot if you'd like, there is also a quit or logout action.

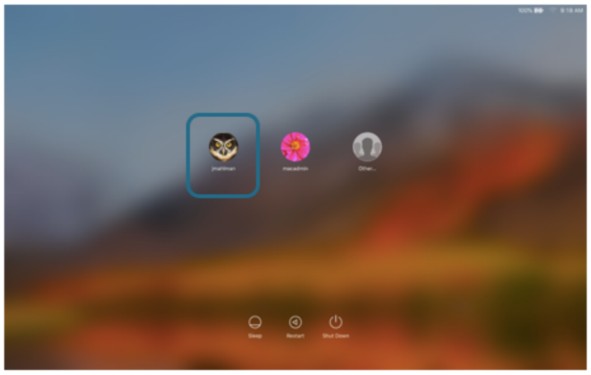
_C_ Right as that happens we clean up the files that we installed (DEPNotify, our logo, and our logs).

_C_ Finally, we create the setup done bom file to tell us that this ran and if the launchdaemon doesn’t unload next it won’t accidentally run provisioning again. And of course we remove our launch daemon.

Note that we’re not unloading it, we had issues with trying to unload it so I decided to just to remove it since the machine was going to reboot anyway.

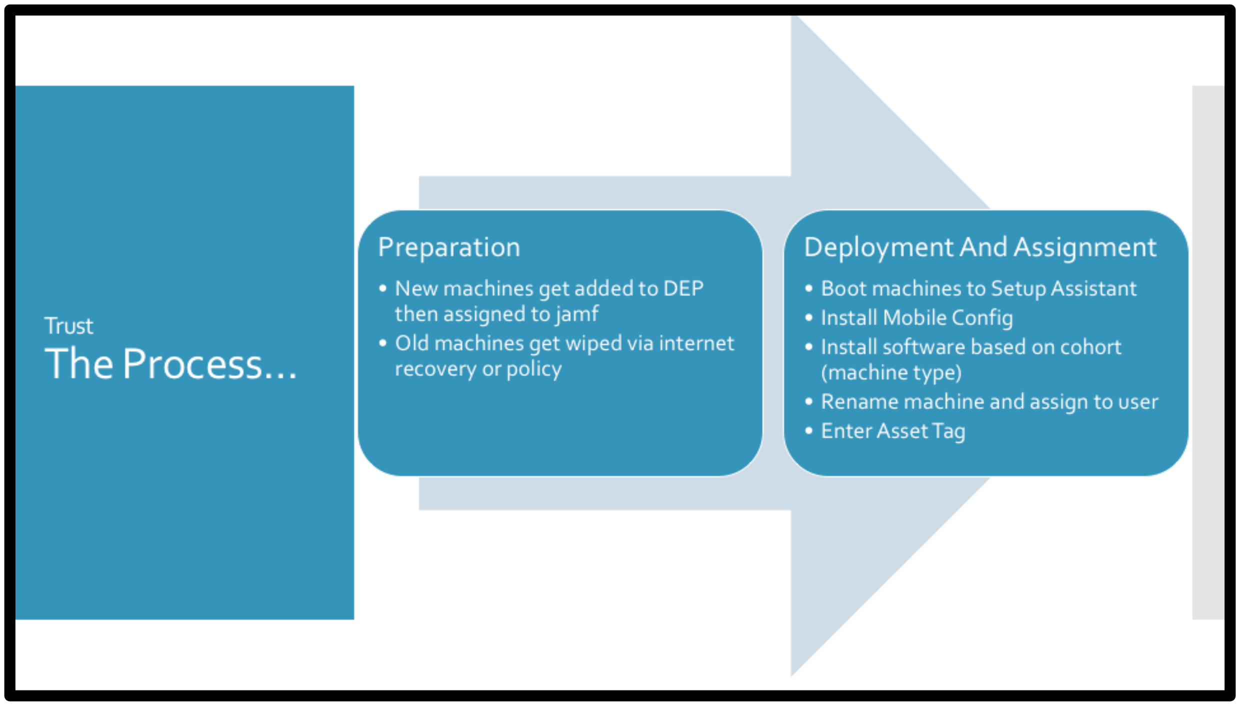


```
    # hold here until the user enters something
    while : ; do
      [[ -f $DNPLIST ]] && break
      sleep 1
    done
    # grab the username from the plist that is created so we can use it to automaticlaly create the account
    USERNAME=$(/usr/libexec/plistbuddy $DNPLIST -c "print 'Assigned User'" | tr [A-Z] [a-z])

    echo "Command: MainTitle: Preparing the system for Deployment" >> $DNLOG
    echo "Command: MainText: Please do not shutdown, reboot, or close your device, it will automatically reboot when complete." >> $DNLOG

    echo "Command: DeterminateManualStep:" >> $DNLOG
    # Do the things! We're calling a single policy now.
    echo "Status: Installing base software..." >> $DNLOG
    $JAMFBIN policy -event enroll-firstRunFACSTAFF

    echo "Command: DeterminateManualStep:" >> $DNLOG
    echo "Status: Creating local user account with password as username..." >> $DNLOG
    /usr/local/jamf/bin/jamf createAccount -username $USERNAME -realname $USERNAME -password $USERNAME -admin

    echo "Command: DeterminateManualStep:" >> $DNLOG
    echo "Status: Assigning and renaming device..." >> $DNLOG
    $JAMFBIN policy -event enroll-assignDevice

    echo "Status: Updating Inventory..." >> $DNLOG
    $JAMFBIN recon

    echo "Command: MainTitle: Almost done!" >> $DNLOG
    echo "Command: DeterminateManualStep:" >> $DNLOG
    echo "Status: Checking for and installing any OS updates..." >> $DNLOG
    /usr/sbin/softwareupdate -ia

    kill "$caffeinatepid"

    echo "Command: RestartNow:" >>  $DNLOG

    # Remove DEPNotify and the logs
    /bin/rm -Rf /var/tmp/DEPNotify.app
    /bin/rm -Rf /var/tmp/uarts-logo.png
    /bin/rm -Rf $DNLOG
    /bin/rm -Rf $DNPLIST

    # Wait a few seconds
    sleep 5
    # Create a bom file that allow this script to stop launching DEPNotify after done
    /usr/bin/touch /var/db/receipts/com.uarts.provisioning.done.bom
    # Remove the Launch Daemon
    /bin/rm -Rf /Library/LaunchDaemons/com.uarts.launch.plist

  fi
  exit 0
```



Reboot complete and

_C_ we now have our new local user and everything installed.



Trust

The Process…

Preparation

- New machines get added to DEP then assigned to jamf
- Old machines get wiped via internet recovery or policy

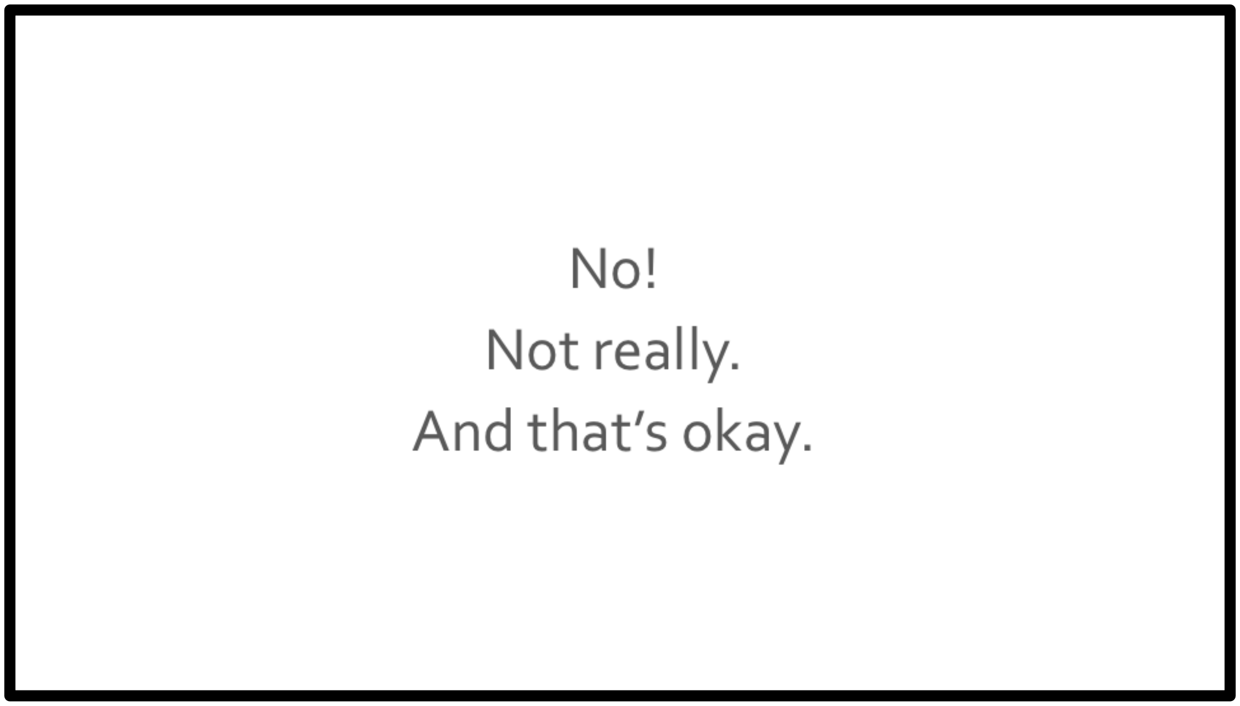
Deployment And Assignment

- Boot machines to Setup Assistant
- Install Mobile Config
- Install software based on cohort (machine type)
- Rename machine and assign to user
- Enter Asset Tag

So there we go, our new process. Two steps..kind of.

So.

Is this imaging?



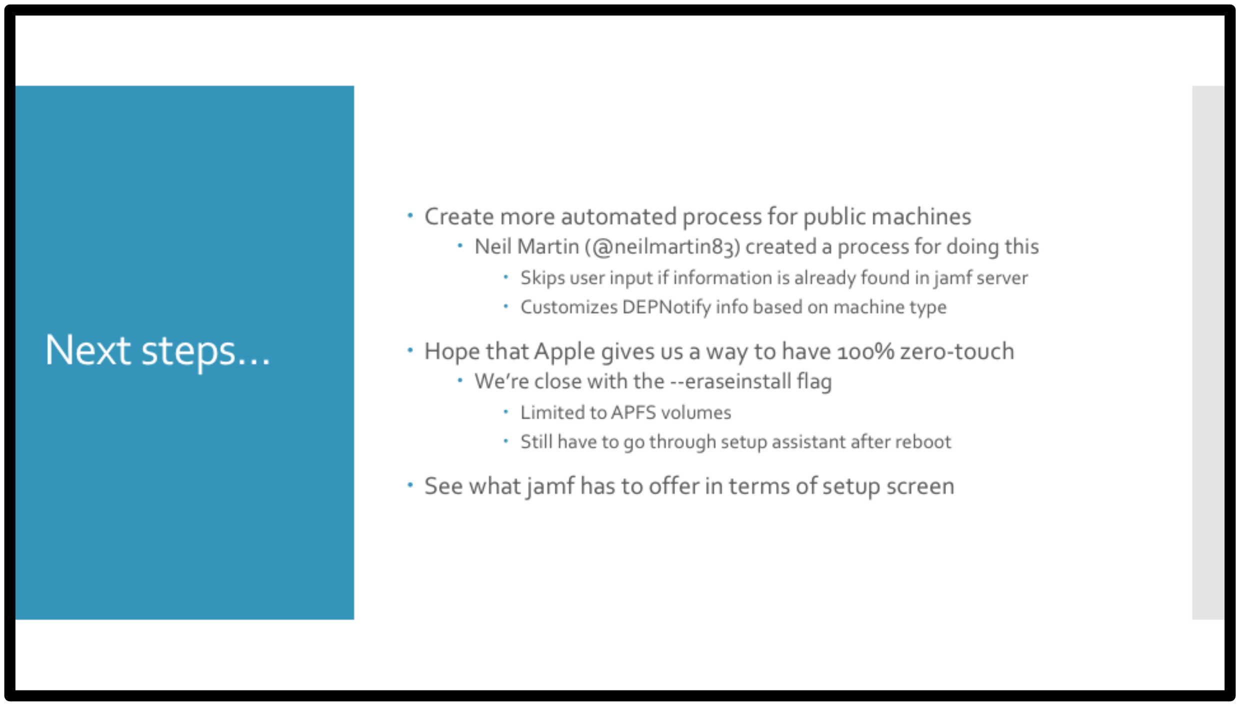
No!

Not really.

And that’s okay.

No, not really.

_C_ And that’s okay because it actually works well for us. At least in this case.


Next steps…

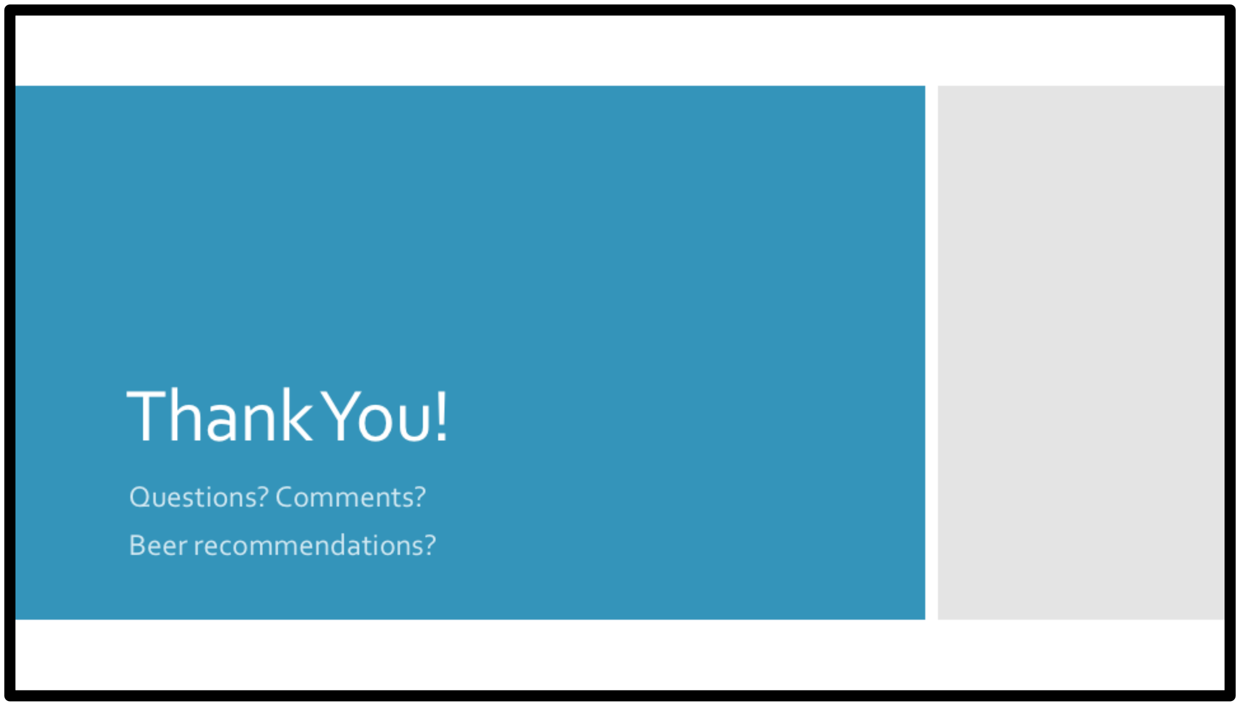
- Create more automated process for public machines
  - Neil Martin (@neilmartin83) created a process for doing this
    - Skips user input if information is already found in jamf server
    - Customizes DEPNotify info based on machine type
- Hope that Apple gives us a way to have 100% zero-touch
  - We're close with the --eraseinstall flag
    - Limited to APFS volumes
    - Still have to go through setup assistant after reboot
- See what jamf has to offer in terms of setup screen

Having success with this we have a lot more things we want to do.

_C_ We're going to create a more automated process.
_C_ Neil Martin has one already for doing this that skips user into if the machine is found in the server and changes DEP info based on machine type

_C_ Hope that apple gives us true zero-touch.
_C_We're getting close with the new erase install flag but that is limited to APFS (which apparently will not work on all HDs in 10.14?) and we still have to go through setup assistant.

_C_ And finally see what jamf has to offer in terms of DEP setup. They're working on something now, I've been trying to get in on every webex they offer me to give input and see what they're doing. So far, I like what I see but I think we're still a bit off from a release.

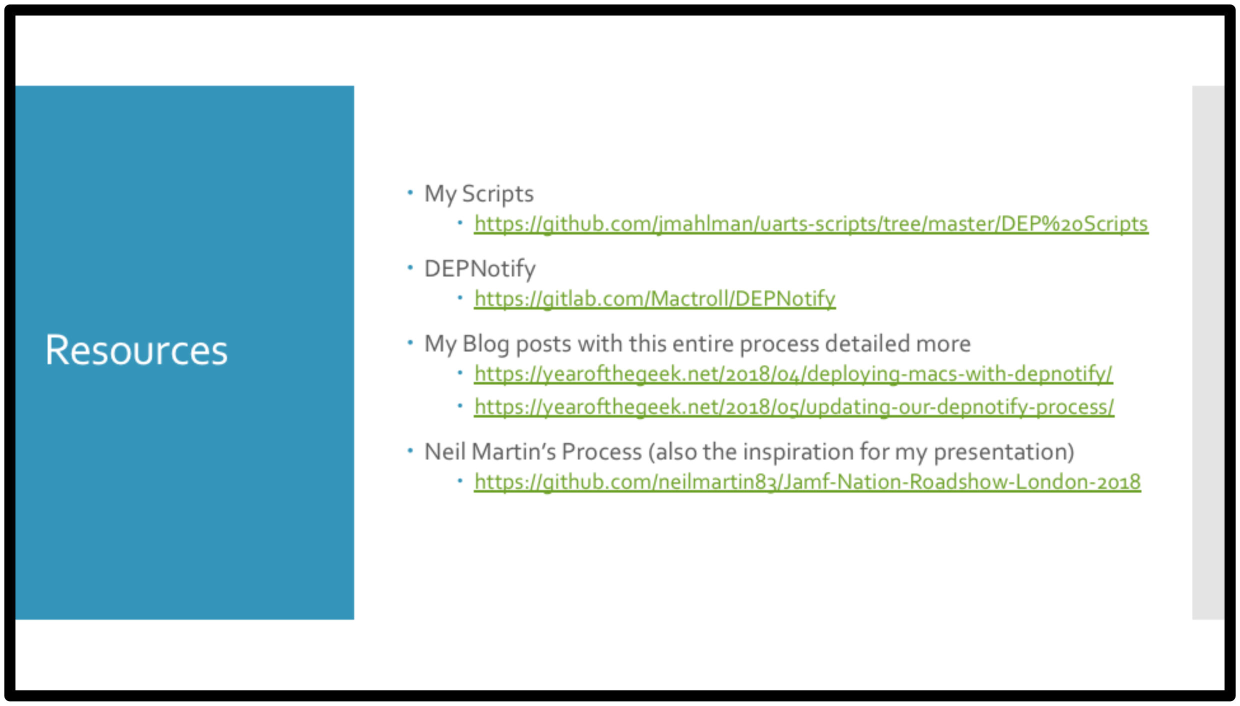


Thank You!

Questions? Comments?

Beer recommendations?

Thanks much!



Resources

- My Scripts
  - https://github.com/jmahlman/uarts-scripts/tree/master/DEP%20Scripts
- DEPNotify
  - https://gitlab.com/Mactroll/DEPNotify
- My Blog posts with this entire process detailed more
  - https://yearofthegeek.net/2018/04/deploying-macs-with-depnotify/
  - https://yearofthegeek.net/2018/05/updating-our-depnotify-process/
- Neil Martin's Process (also the inspiration for my presentation)
  - https://github.com/neilmartin83/Jamf-Nation-Roadshow-London-2018

Some resources for this presentation including my scripts and my blog post on this entire process.

I'm also including a link to Neil Martin's process because I mentioned it and it's probably very useful for others.