Going MAD: From closed in box to ready to rock

Installing, configuring and combining Munki, Autopkg, DeployStudio and other tools to prepare new machines for end users.

Steve Yuroff
@swy

what are we here to learn?

I intend to explain from the ground up why each tool exists, how they can work together make your life better

This presentation is going to acknowledge the existence of many different tools: You don't need to write down URLs: they are all included in the final slides, which will be available for download from the PSUMac website.

what are we here to learn?

- why each tool exists: what purpose they serve and how they will help you

I intend to explain from the ground up why each tool exists, how they can work together make your life better
This presentation is going to acknowledge the existence of many different tools: You don't need to write down URLs: they are all included in the final slides, which will be available for download from the PSUMac website.

what are we here to learn?

- why each tool exists: what purpose they serve and how they will help you
- how to obtain and install each, and do basic configuration

I intend to explain from the ground up why each tool exists, how they can work together make your life better
This presentation is going to acknowledge the existence of many different tools: You don't need to write down URLs: they are all included in the final slides, which will be available for download from the PSUMac website.

what are we here to learn?

- why each tool exists: what purpose they serve and how they will help you
- how to obtain and install each, and do basic configuration
- how to connect them to make an automated process which yields a computer configured with current software, installed without applying a master image

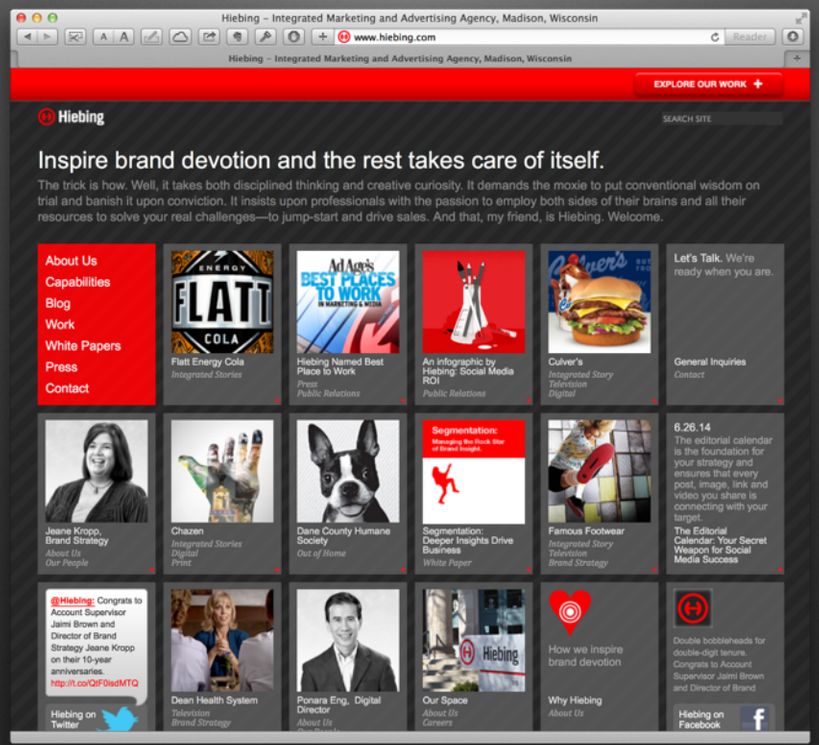
I intend to explain from the ground up why each tool exists, how they can work together make your life better
This presentation is going to acknowledge the existence of many different tools: You don't need to write down URLs: they are all included in the final slides, which will be available for download from the PSUMac website.

Who are you listening to?

- Steve Yuroff: @swy

For last 13 years, I've been the whole IT department for a marketing and advertising agency named Hiebing, out of Madison, WI which means I handle not just deploying macs, but manage the network, the virtualization infrastructure, the Windows and LAMP servers on it, every possible level of support, ect What you'll see is crafted around my needs, to deploy and maintain machines for a 75 person creative agency.

Who are you listening to?



For last 13 years, I've been the whole IT department for a marketing and advertising agency named Hiebing, out of Madison, WI which means I handle not just deploying macs, but manage the network, the virtualization infrastructure, the Windows and LAMP servers on it, every possible level of support, ect What you'll see is crafted around my needs, to deploy and maintain machines for a 75 person creative agency.

Who are you?

Who are you?

- Not experienced in deploying OS X to end users.

Who are you?

- Not experienced in deploying OS X to end users.
- Experienced in deploying OS X to end users, but exploring alternatives to monolithic imaging or commercial products.

Who are you?

- Not experienced in deploying OS X to end users.
- Experienced in deploying OS X to end users, but exploring alternatives to monolithic imaging or commercial products.
- Somebody who has done this, and wants to see how someone else has handled it.

The main ingredients

OS X Server - Why?

Will provide:

Webserver for munki

NetInstall (a/k/a Netboot)

Potentially file sharing services

The main ingredients

OS X Server - Why?

Will provide:

Webserver for munki

NetInstall (a/k/a Netboot)

Potentially file sharing services

The main ingredients

An open source system to automatically install software on OS X computers

Consists of 2 parts

The files on a web server

Software that runs on each managed computer, which queries that web server periodically, asking if there's new software for it to install, and if so, doing that.

An open source system to automatically install software on OS X computers

Consists of 2 parts

The files on a web server

Software that runs on each managed computer, which queries that web server periodically, asking if there's new software for it to install, and if so, doing that.

The main ingredients

AutoPkg solves two problems:

•Not all software comes from the vendor in pkg format, ready to install. Often need to be tweaked to fix bad installers, with version number oddities, assumptions that a GUI user is installing the app and breaking at loginwindow and other problems

•Even if the provided installers are perfect, keeping munki fed with the latest version of an application has admin overhead. Autopkg can reduce the process of keeping most of a munki repository up to date down to a single command- which can be automated.

"Recipes" are the driving force behind Autopkg. Each recipe is just that: a series of steps autopkg will take to download installers and adjust as needed to output a pkg that can be imported into munki, ready for installation on clients. Recipes come from community contribution

The main ingredients

```
steyur-c02ll46u:~ syuroff$ autopkg
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 informatation 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
steyur-c02ll46u:~ syuroff$
```

AutoPkg solves two problems:

•Not all software comes from the vendor in pkg format, ready to install. Often need to be tweaked to fix bad installers, with version number oddities, assumptions that a GUI user is installing the app and breaking at loginwindow and other problems

•Even if the provided installers are perfect, keeping munki fed with the latest version of an application has admin overhead. Autopkg can reduce the process of keeping most of a munki repository up to date down to a single command- which can be automated.

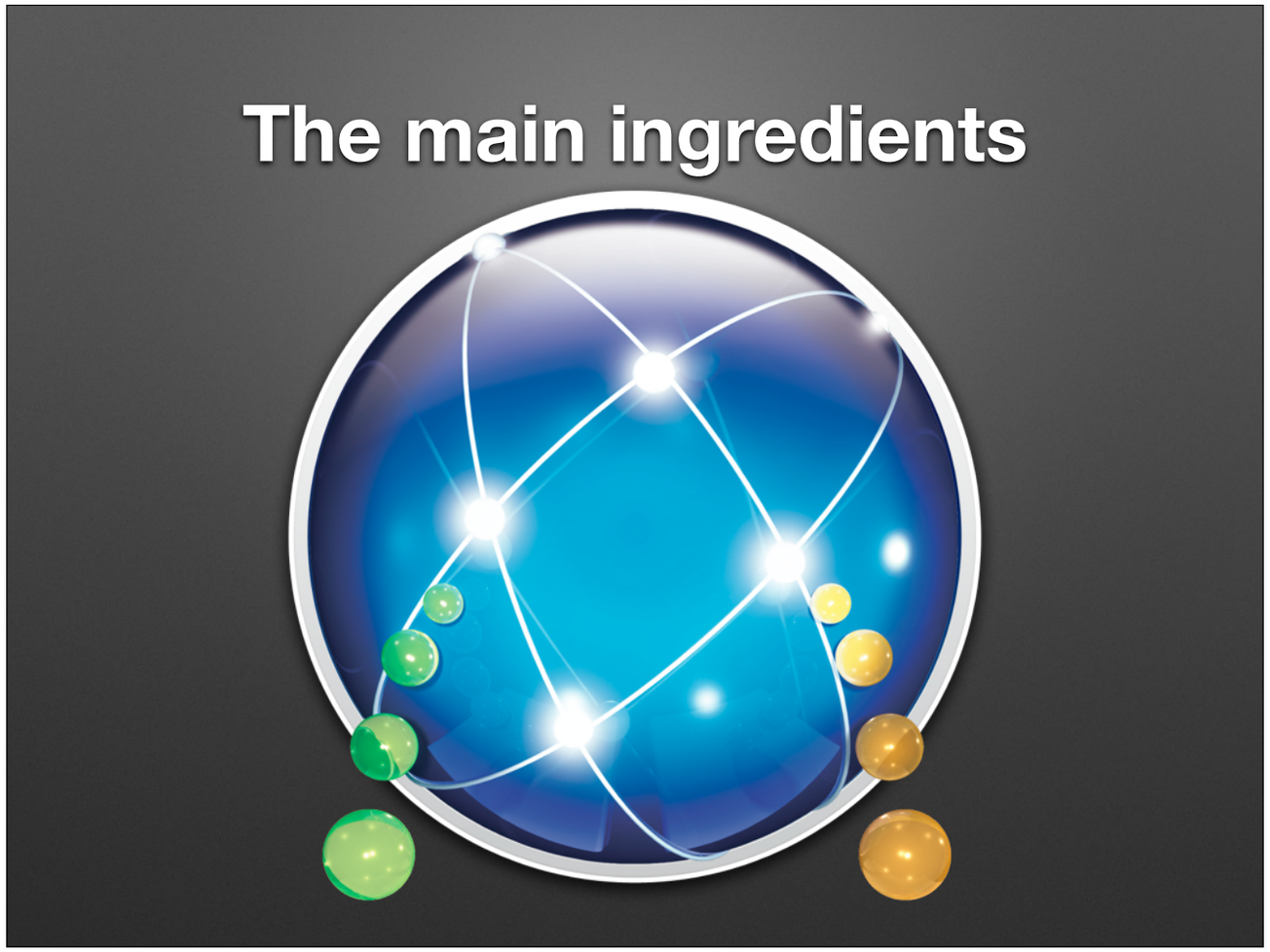
"Recipes" are the driving force behind Autopkg. Each recipe is just that: a series of steps autopkg will take to download installers and adjust as needed to output a pkg that can be imported into munki, ready for installation on clients. Recipes come from community contribution

The main ingredients

DeployStudio has 2 parts:

1) A service, running on a local server. The main purpose of this service (for this session) is to provide *workflows*, a series of steps one can apply to a computer running the Deploystudio Runtime.

2) A netboot disk image we'll build, which is how a computer runs the DeployStudio runtime, and leaves the internal volume free to install software, documents, profiles, scripts and more.



DeployStudio has 2 parts:

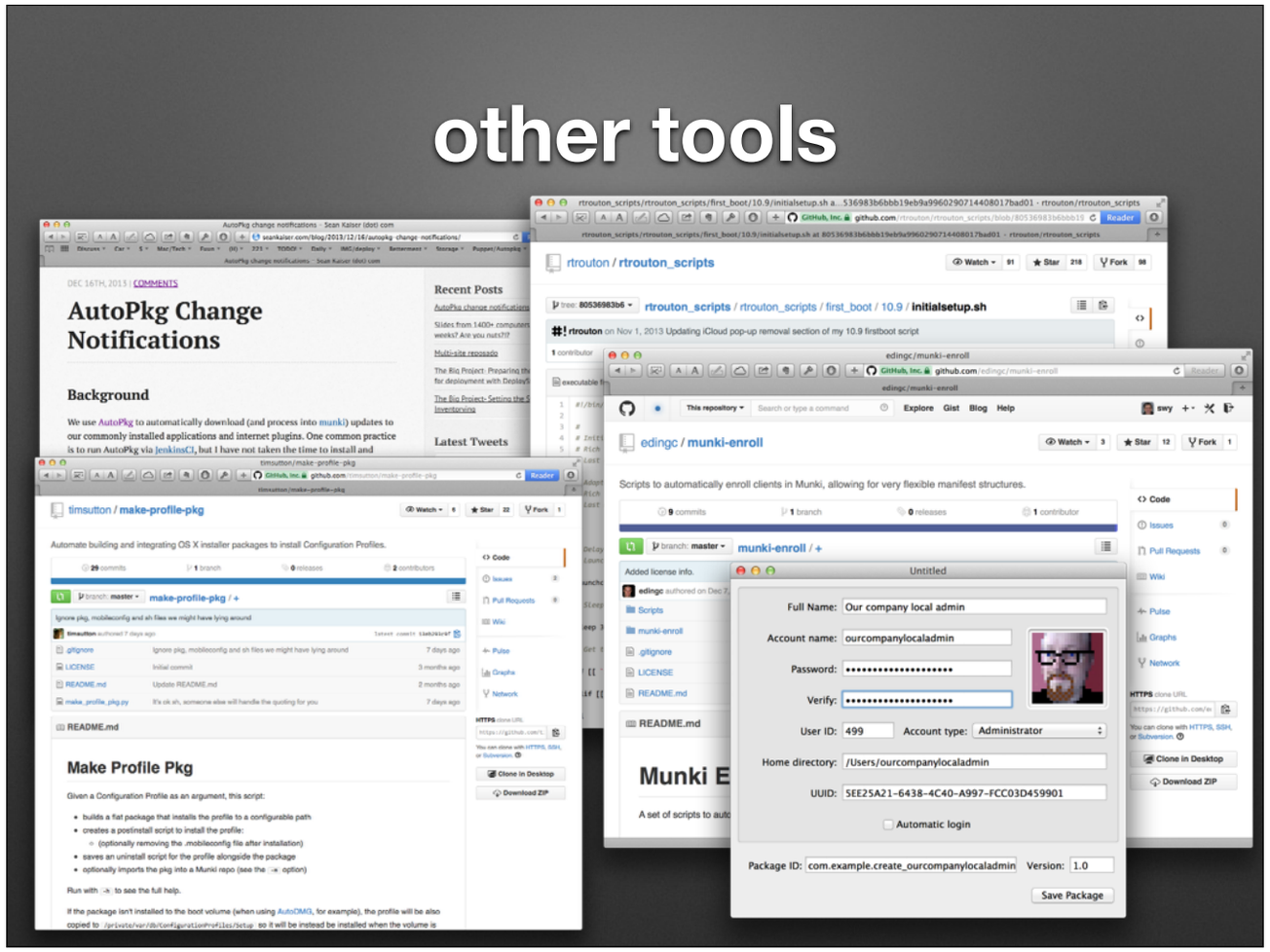
1) A service, running on a local server. The main purpose of this service (for this session) is to provide *workflows*, a series of steps one can apply to a computer running the Deploystudio Runtime.

2) A netboot disk image we'll build, which is how a computer runs the DeployStudio runtime, and leaves the internal volume free to install software, documents, profiles, scripts and more.

other tools

And a collection of other tools to make it all possible.

other tools



And a collection of other tools to make it all possible.

Download & install DS

Download installer from DeployStudio's website
Run installer on any mac you want to be the DeployStudio server. In my office, that's a Mac Mini, which also runs munki and netboot services.

I'll presume you can run a pkg installer without me- accept all defaults

Download & install DS

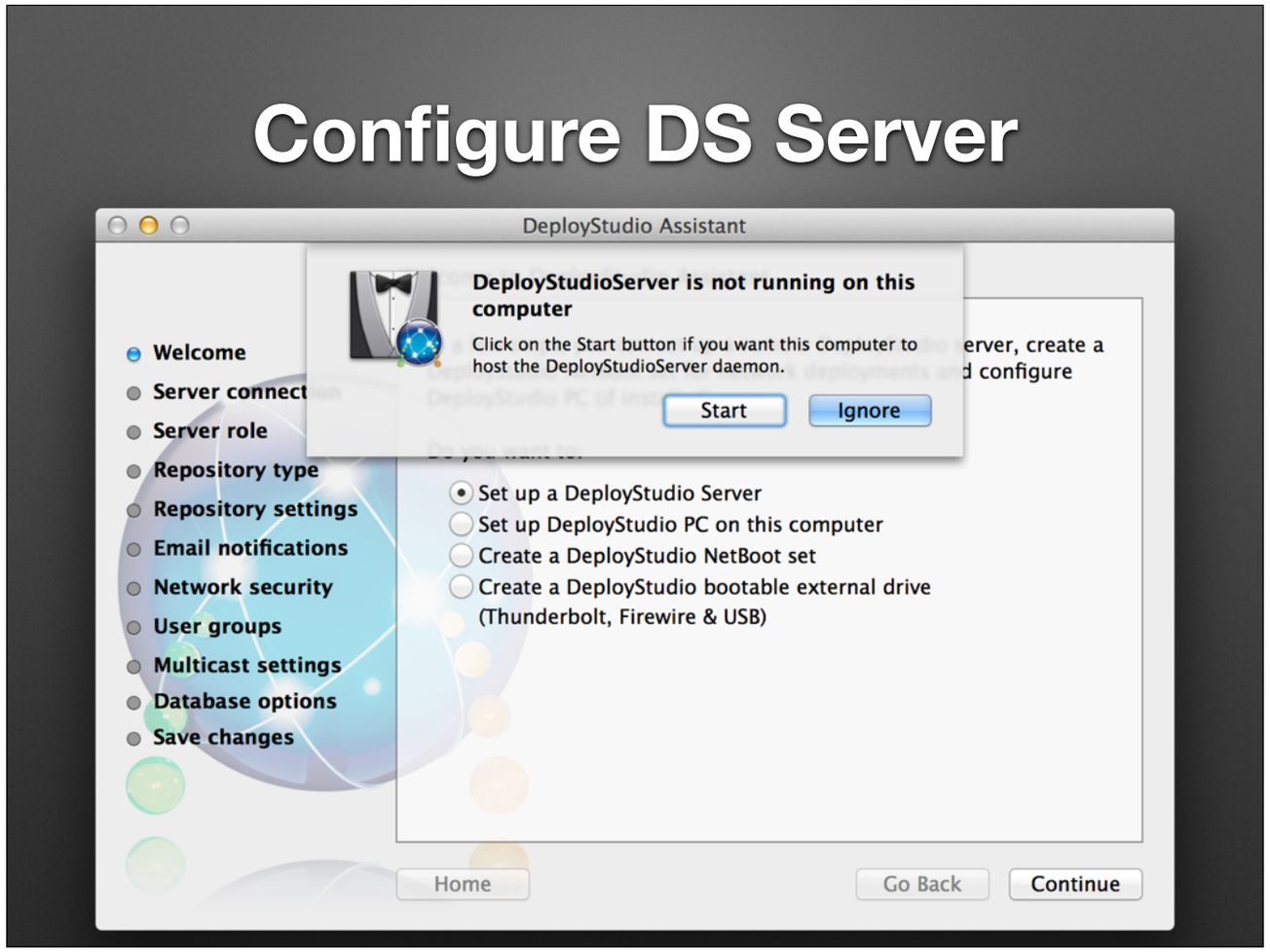
Download installer from DeployStudio's website
Run installer on any mac you want to be the DeployStudio server. In my office, that's a Mac Mini, which also runs munki and netboot services.

I'll presume you can run a pkg installer without me- accept all defaults

Configure DeployStudio

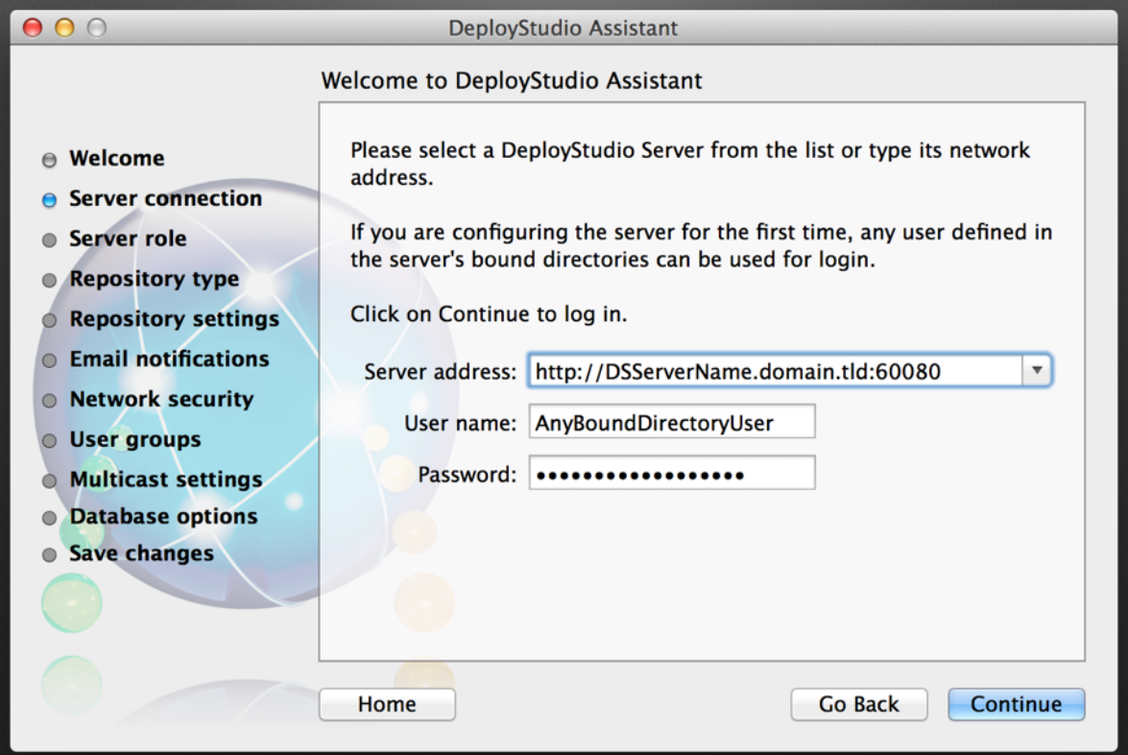
Once installed, launch the DeployStudio Assistant from the Utilities folder

Configure DS Server



We will start the service on this machine, and continue with the default "Set up a DeployStudio Server" option.

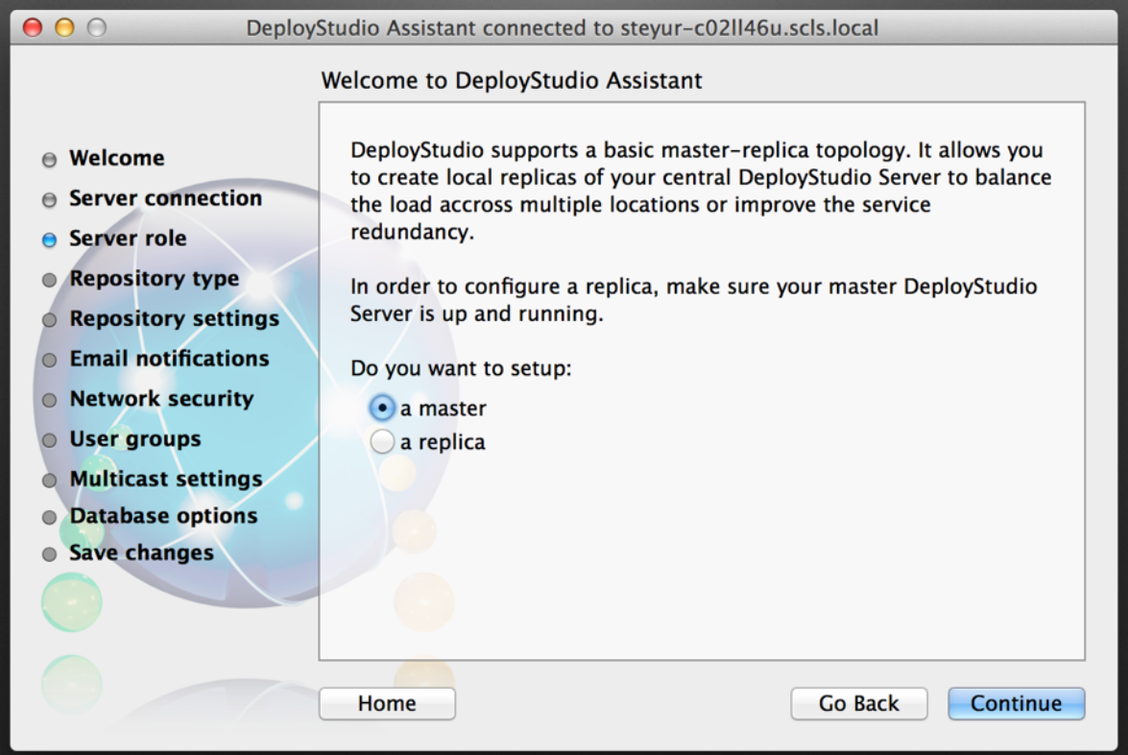
Configure DS Server



You will need the assistance your DNS admins (or yourself) to get your server name defined in DNS. **Pro tip:** I recommend you name your services as aliases and give machines real names: this makes it easier to move and relocate services to new hardware, vs renaming hardware.

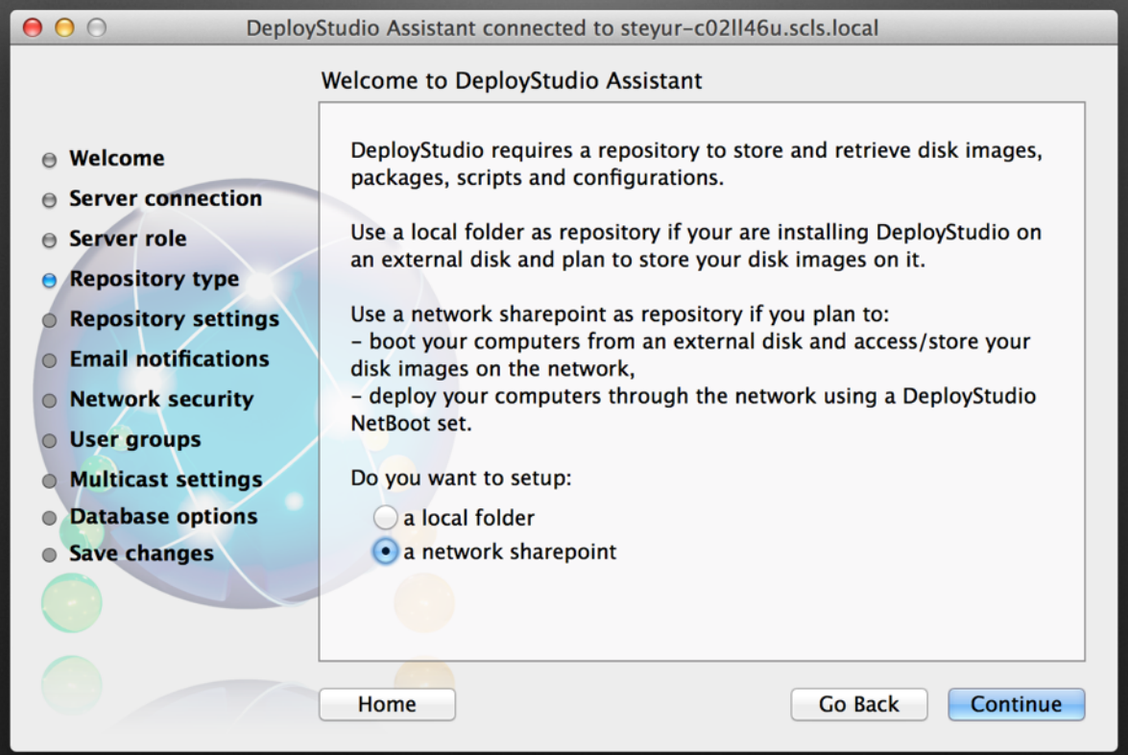
Any user from a bound directory can be used to authenticate

Configure DS Server



This is the first in this environment: it's a Master

Configure DS Server

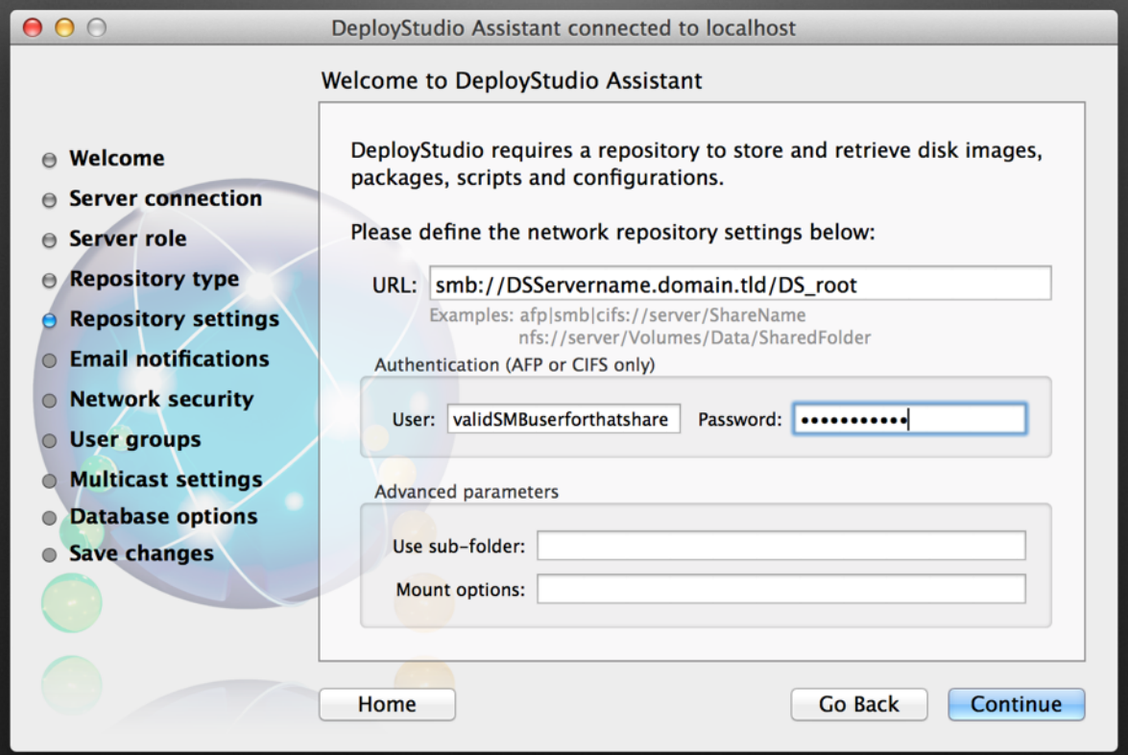


This question is "where are we going to store the Deploystudio stuff?"

Local folder: Right choice if you're booting your computers from a dedicated local disk- USB/Thunderbolt.

We're going to use NetBoot for this project, so "a network sharepoint" is our choice.

Configure DS Server



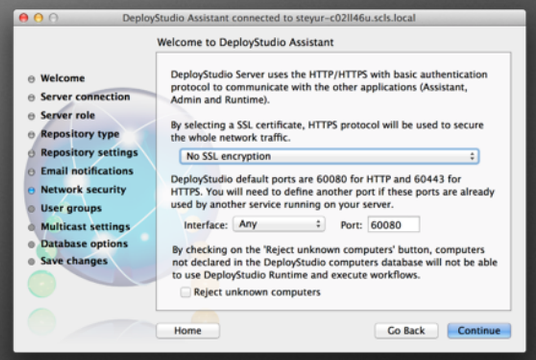
Deploystudio needs a file server to hold all the things it can offer. I'm going to guess that you have an administrative file server where you can store DeployStudio stuff.

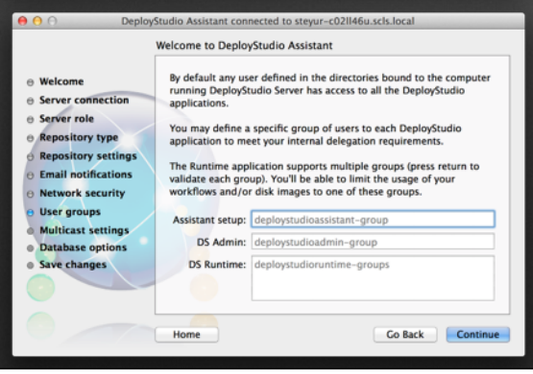
So what kind of size are we talking here? Like anything, it depends. But always easier to have a share with too much space than too little. To replicate what I'm showing here would only take under a gig.

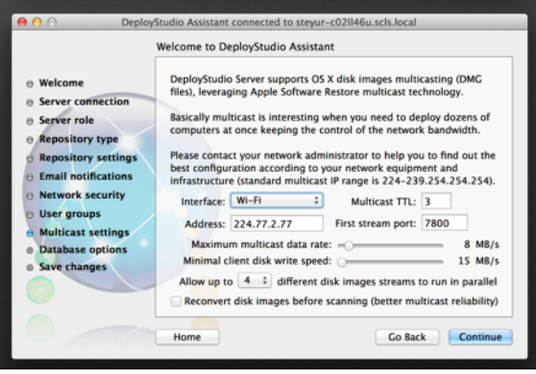
Enter an admin account to read and write from the service here

This means your answer here must be a valid share because your login will be confirmed before you can continue.

Configure DS Server



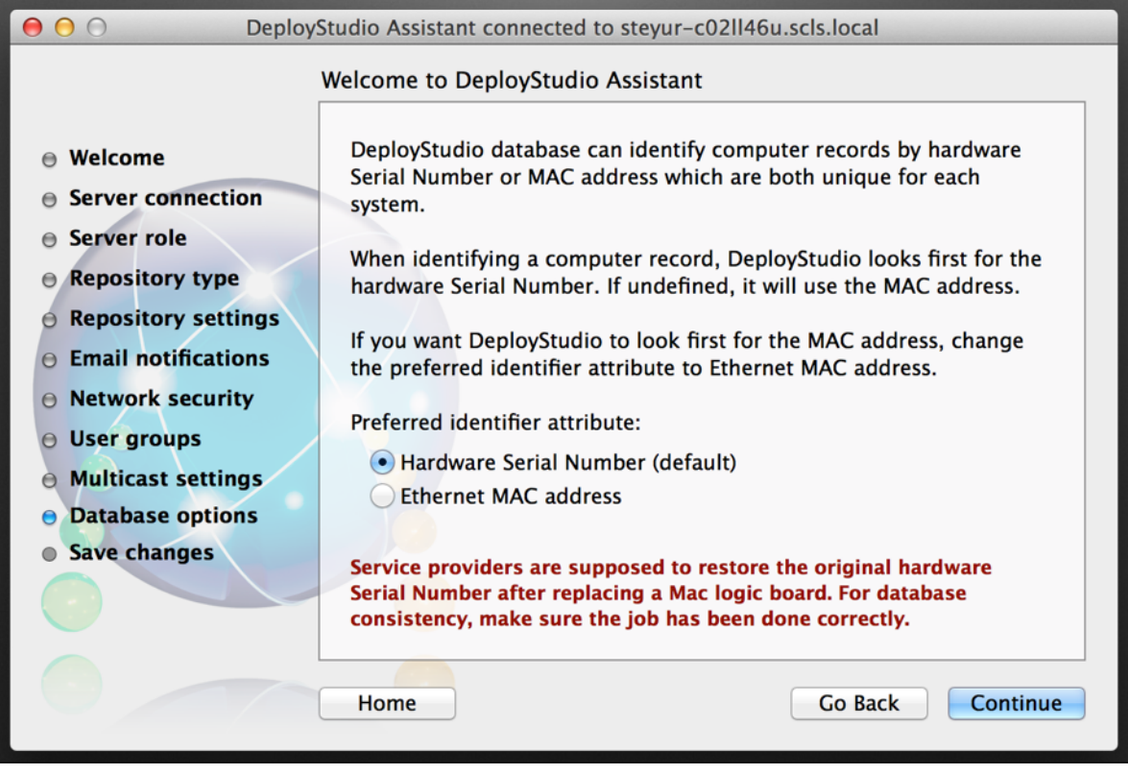




optional settings

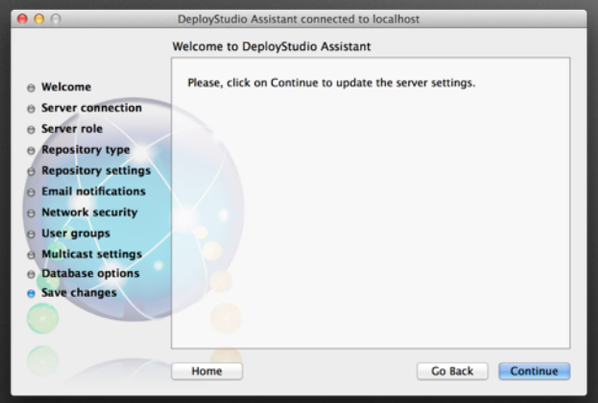
- Email alerts
- SSL
- Limit who can run different tools
- multicast options

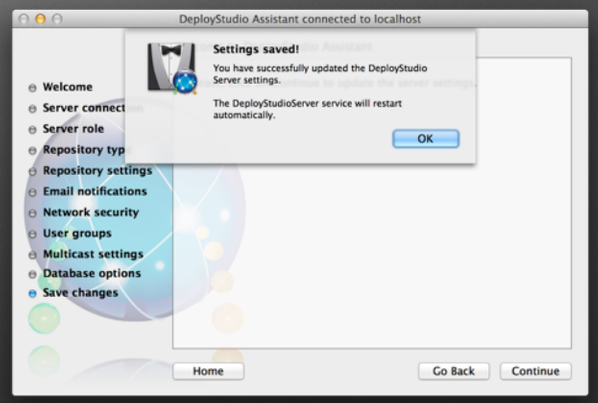
Configure DS Server



Keep with default of hardware serial, espicially if you reuse Ethernet adapters: every computer will look like the same computer if you don't.

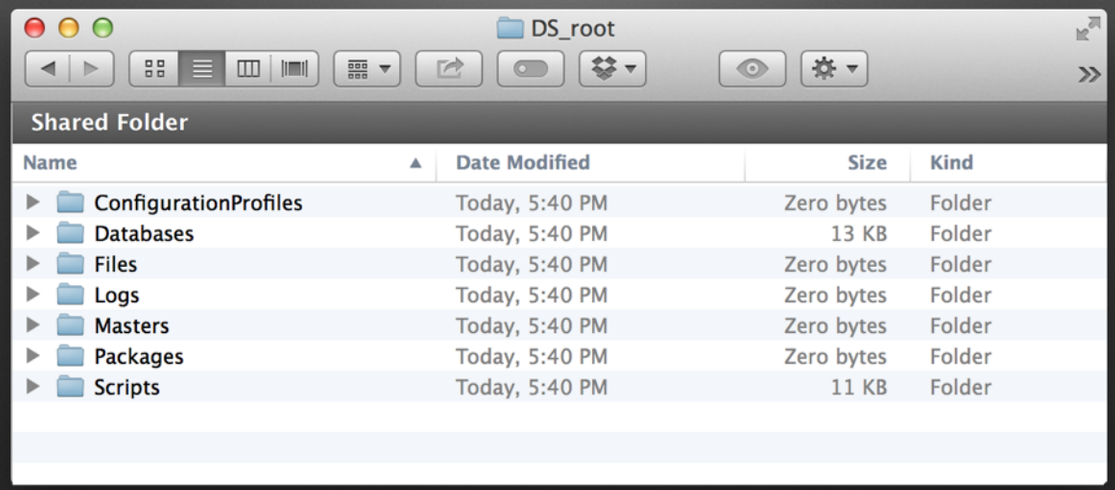
Configure DS Server





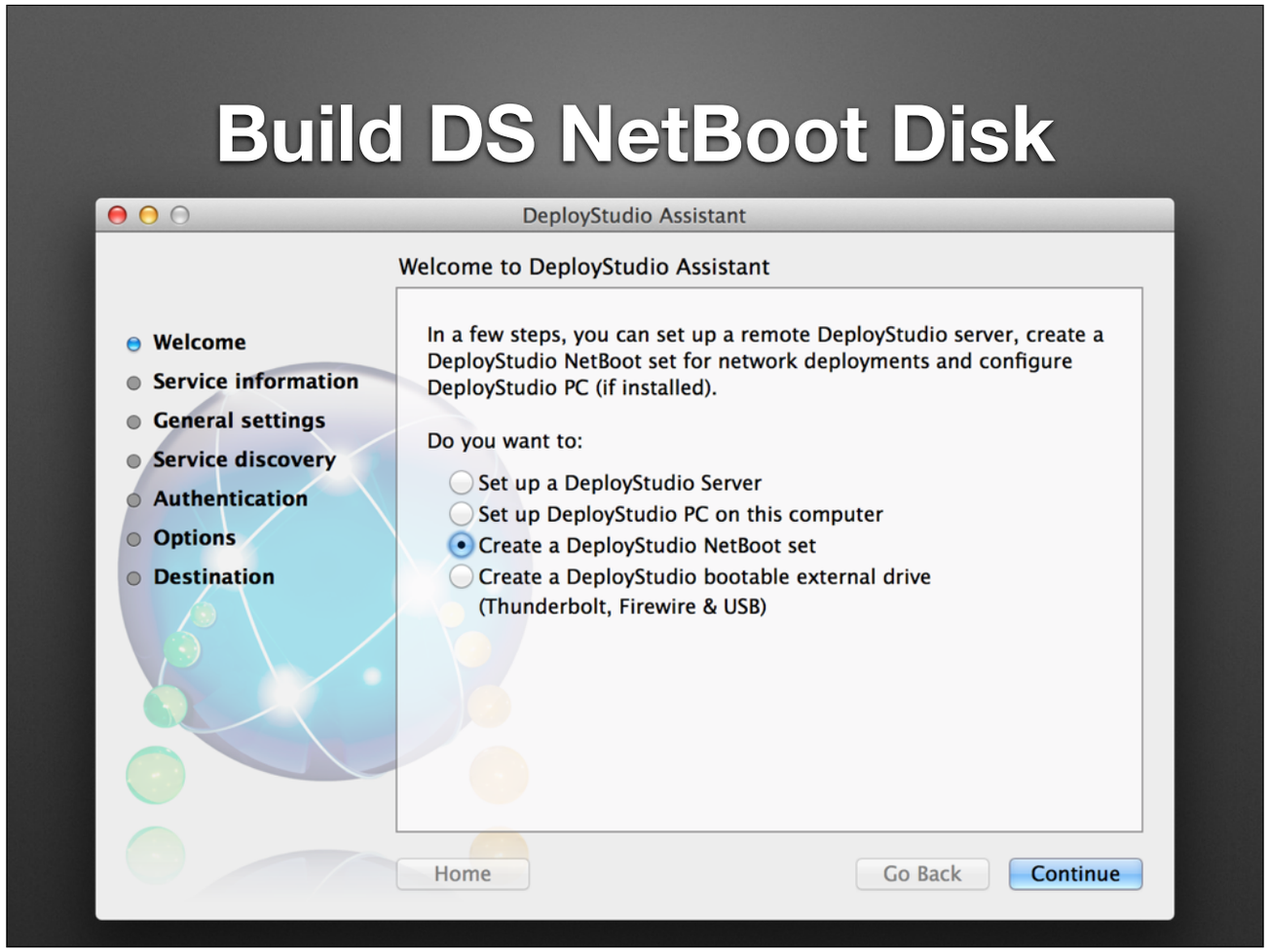
The final 2 clicks are pretty self explanatory, and will bring us back to the assistant main page. At this point, you've configured the DS server, and that new service is running on your server.

folders in DS_root share

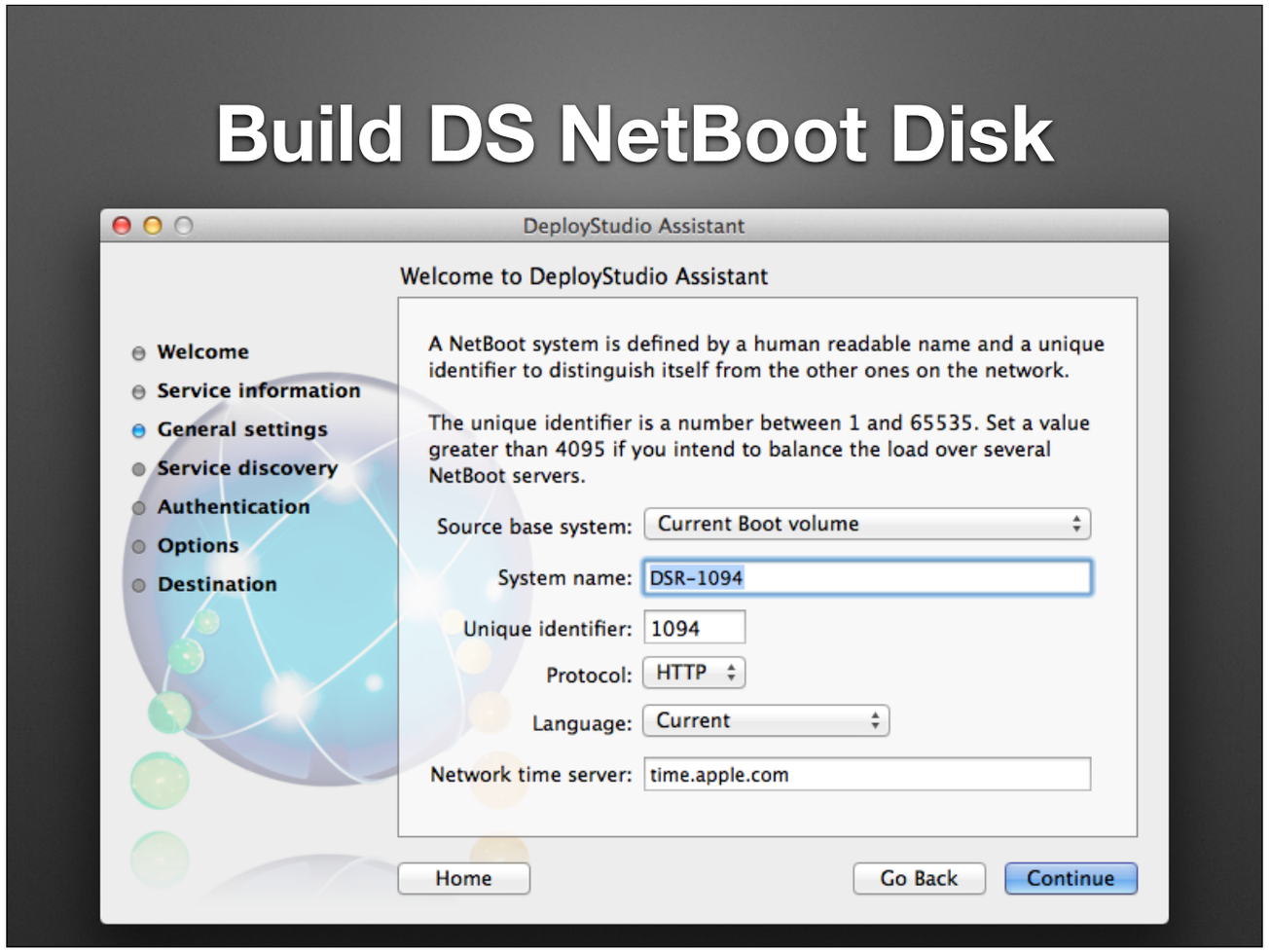


On the 5th screen of the wizard, we declared a **network repository** shared folder as the root of our DeployStudio install. Within that folder, the installer has made these 7 subfolders. These are the folders where DeployStudio's different categories of assets are held. Later on, we'll be returning to the Files folder.

Build DS NetBoot Disk



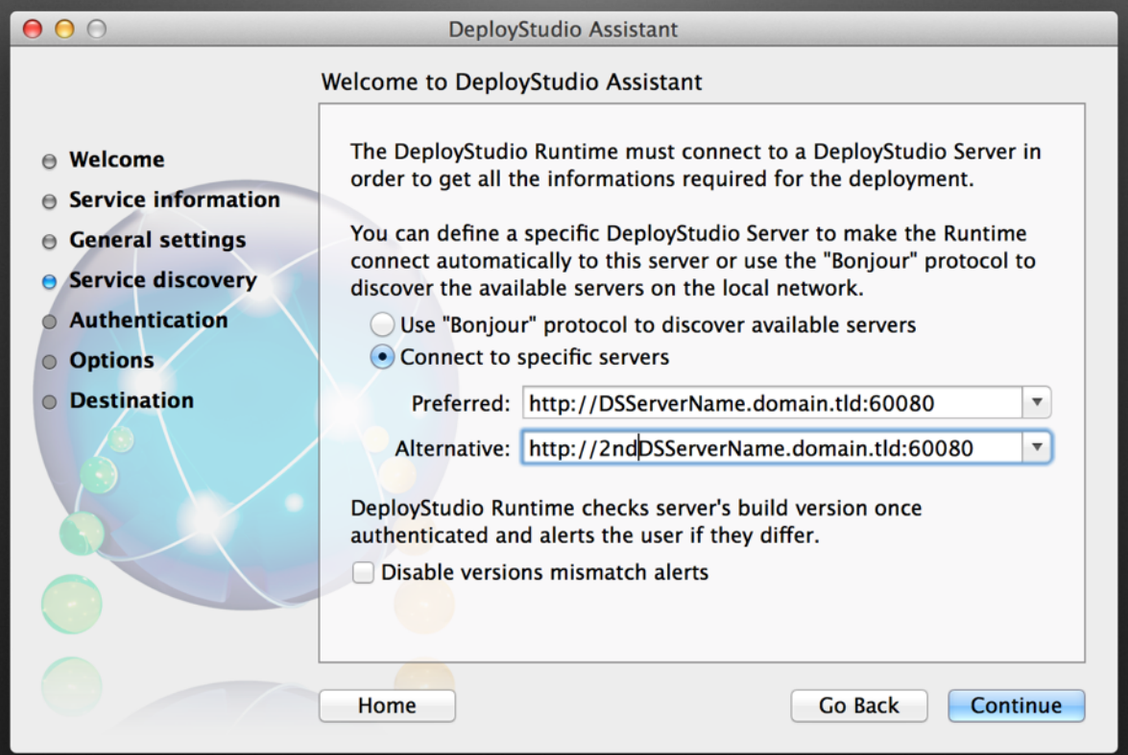
Next step is to create a disk image (called a NBI, for NetBoot Image), which the assistant will make by reading the OS from a local Volume, and combine the DeployStudio runtime app with it, making a bootable disk where loginwindow and Finder are replaced by DeployStudio runtime app. We'l then hand that to OS X Server to provide as a network bootable volume, and boot from it to talk to our DeployStudio server.



It's important that the **Source Base System** is the most current release of OS X possible, though I stay away from the beta releases.

System name is what is shown in Startup Disk Preference Pane, or when found via holding down option at boot.

Build DS NetBoot Disk



We only have 1 server, so Alternative is pointless right now. But if/when you have redundancy, there's the spot for it.

Of course, we're pointing it to the DeployStudio server we just configured.

Build DS NetBoot Disk

If you want to netboot a machine and log into the DS server automatically, fill in the top 2 credentials. This account could be a network or local account on the DS server.

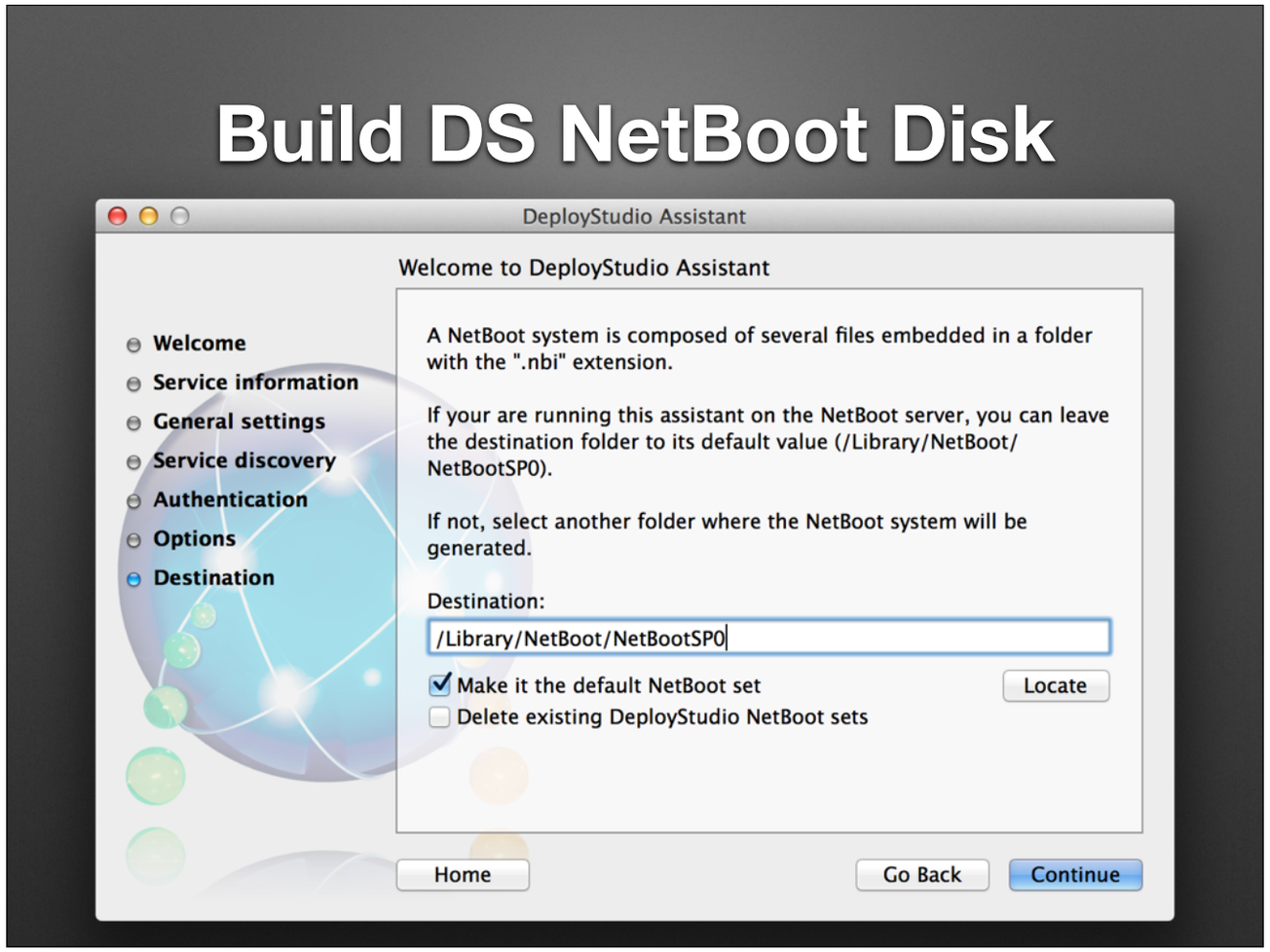
The second set of credentials allow for a remote VNC session to a machine netbooted to a DeployStudio disk.

I like to add in the Display Runtime Log by default option

Build DS NetBoot Disk

It's useful to set a custom image and text- visual verification of which source you're booted from. The cat with a camera is completely optional- cat is not default.

Build DS NetBoot Disk



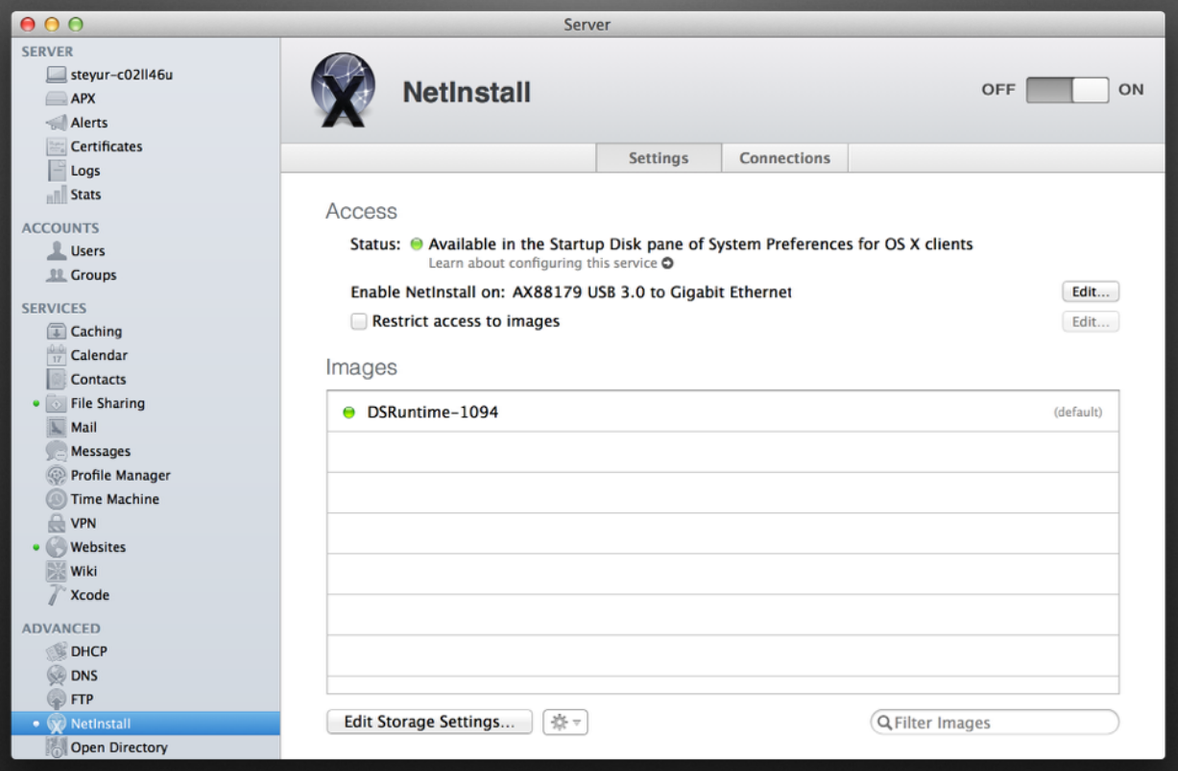
If you enable default NetBoot set, can boot to it by holding N. This may or may not be right for you.
/Library/NetBoot/NetBootSP0 is the default location for NetBoot sets.
This will run for a few minutes, and alert when it's done.
We have now created a DS server and prepared a network disk to boot a new mac from, which is configured to talk to that DS server.
We are ready to move onto other parts before returning to complete our deploystudio configuration. **"but first, any questions?"**

Of course, this comes from the App Store, and installs Server.app in the Applications folder.

OS X Server

Of course, this comes from the App Store, and installs Server.app in the Applications folder.

OS X Server: NetInstall

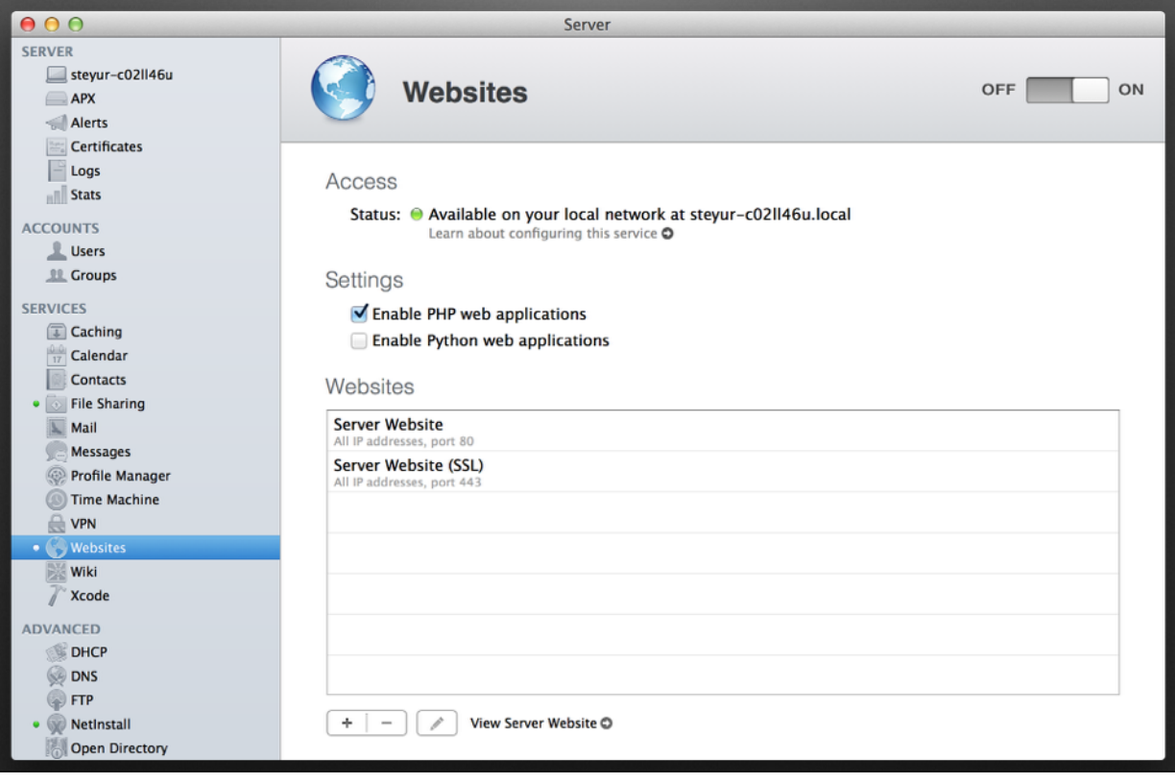


NetInstall hides under advanced

You should only need to turn it on at the top right, as we already put the Deploystudio boot disk in the expected path.

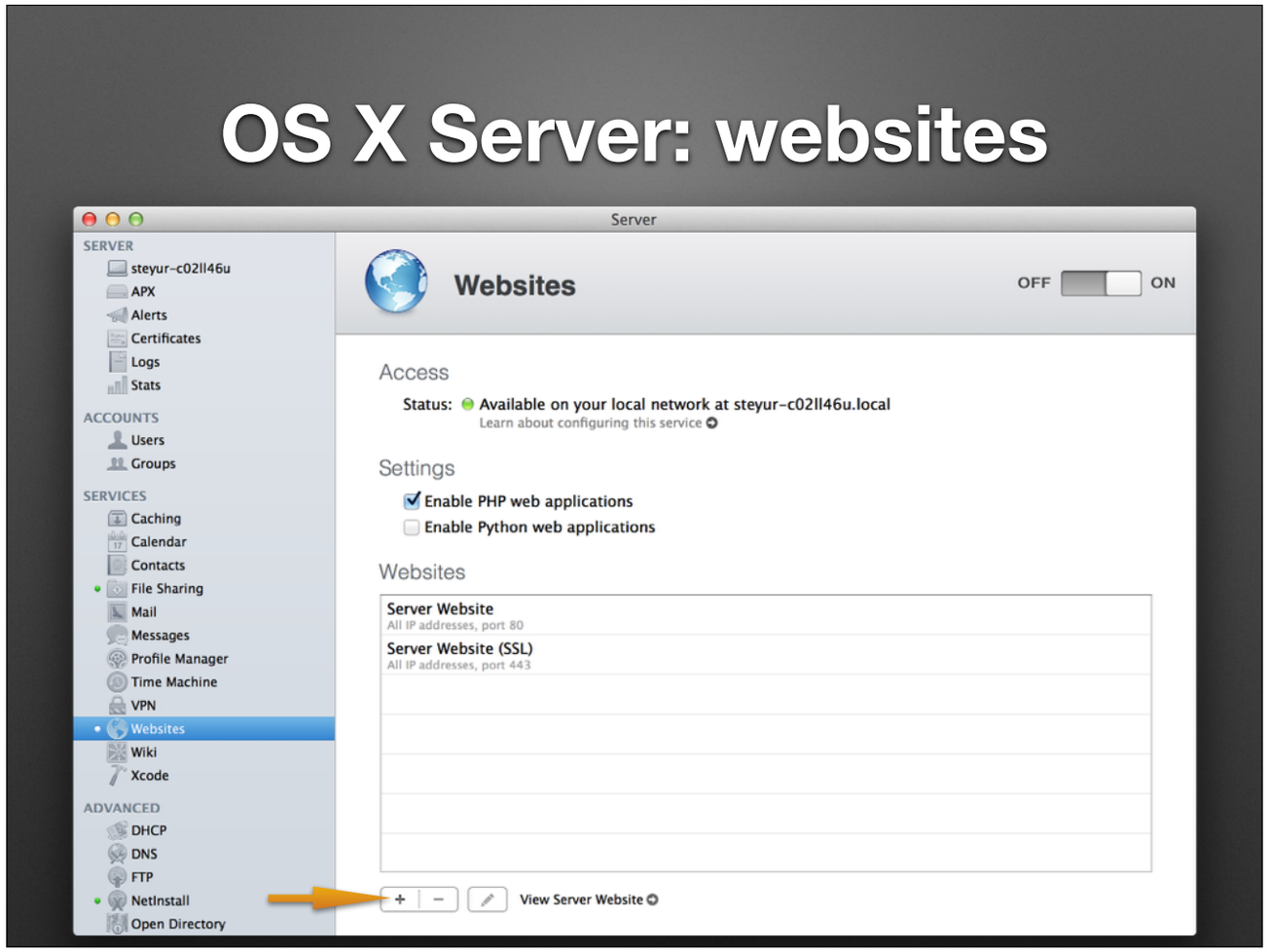
Note your interface: it'll default to Ethernet

OS X Server: websites



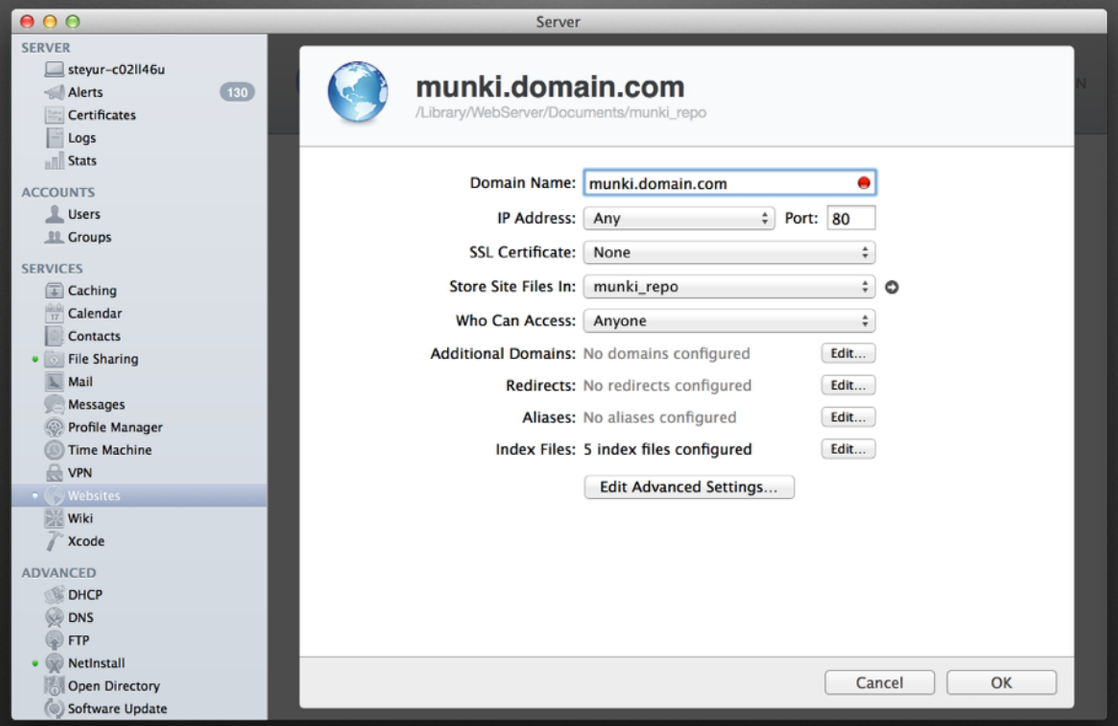
Under Websites, turn it on, enabled PHP web applications, and add a new website

OS X Server: websites



Under Websites, turn it on, enabled PHP web applications, and add a new website

OS X Server: websites

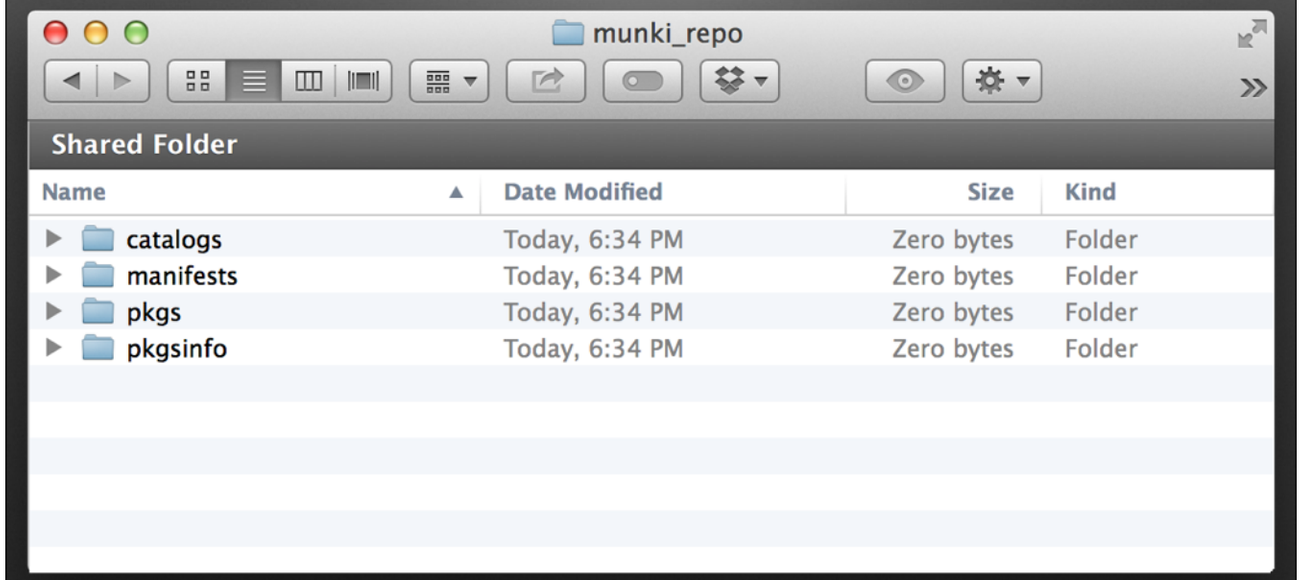


Give it a useful name, and choose a place to hold your munki content, which we'll store in a folder named munki_repo.

Where you store your munki repo is relatively important- depending on how much software you put here and how often you purge versions, it can grow large. Choose a destination with space for growth. My munki repo is now 26gigs

Of course, an appropriate variation of munki.domain.com will be needed for you, with DNS

Required munki folders



Now that we have the base for our munki repository set, there are 4 specific folders we need to make. Do this in the Finder or via Terminal, but the names must match these:

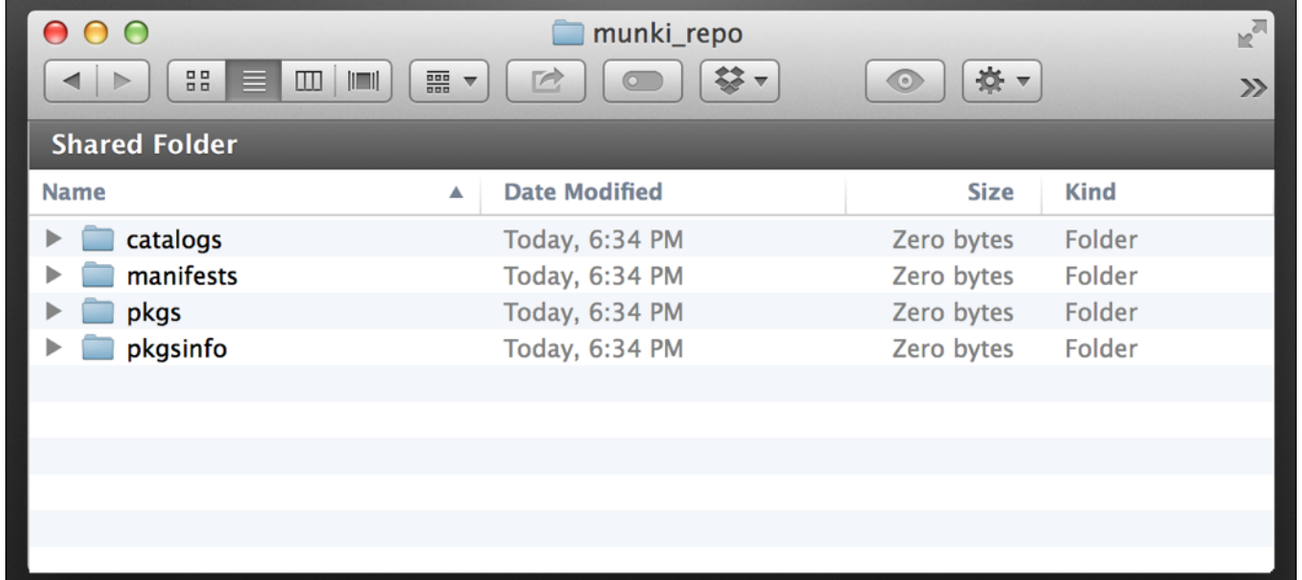
catalogs: a list of software that could be offered for installation. A munki server typically has at a minimum of 2 catalogs: one holding software versions that are being tested, and one for the approved releases.

manifests: a list of what gets installed on that client, selected from the catalogs. It's the filled out order form in a real catalog.

pkgs: the installers will be stored here

pkgsinfo: holds the metadata about the installers

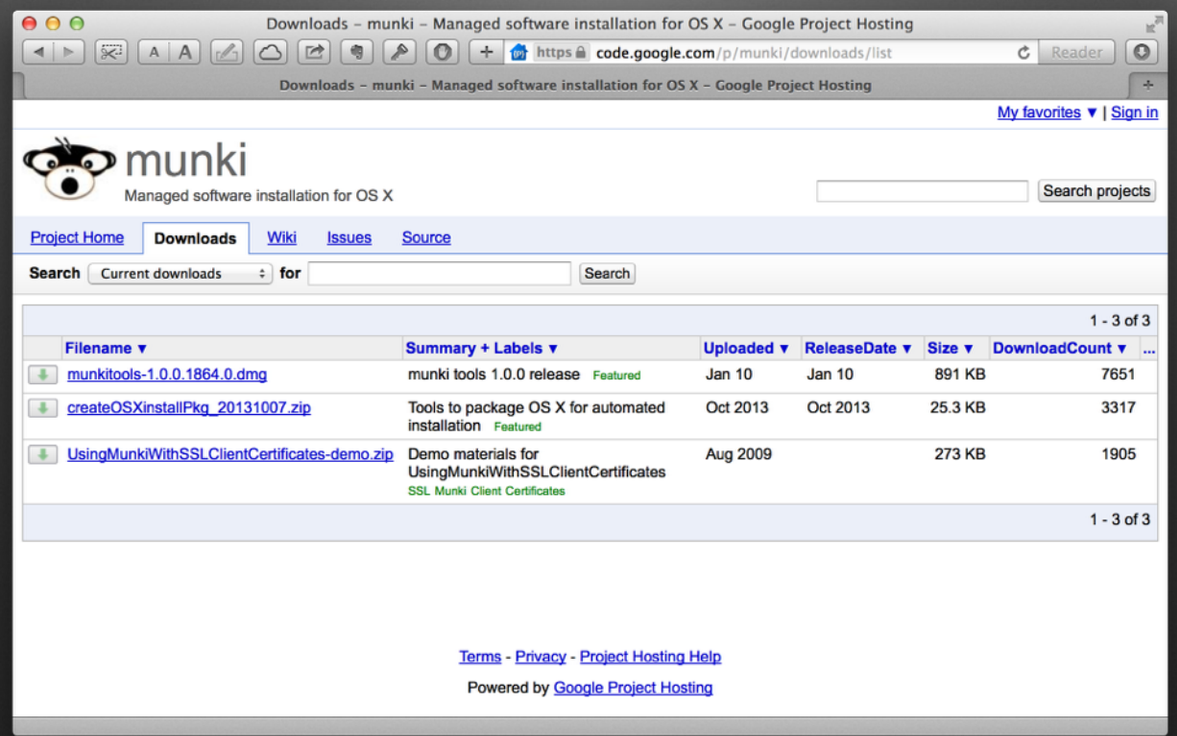
Required munki folders



REVIEW:

At this point, we've configured our two OS X server services of NetInstall and Web services, and set up the required folder structure for munki. We now need to install and configure the tools that manage a munki repository.

Munki tools



Munki tools come from the Google Code website

Tools are to facilitate importing and managing content in the munki repository- the server is just Apache

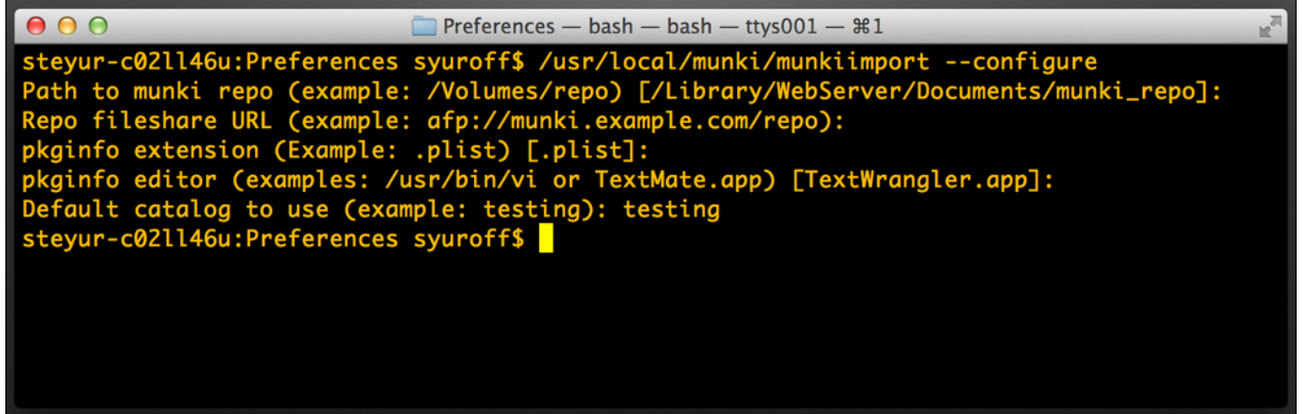
Standard .pkg install- I again trust you to run that installer.

The tools this installer provides are saved to /usr/local/munki

They are all Python scripts

Tools will need to be installed on server and clients. (We'll turn back to DeployStudio to put them on clients)

configure munkiimport



munkiimport is a command line tool (it's one of the Python scripts) for importing software into our munki repo. It was installed with the munki tools, and now we need to tell it how our munki repository is configured, and what tools we prefer to edit text with. *walk through each line*

Path: where is that website you're serving?

Repo fileshare: relevant if you're importing to a remote machine. We're not.

pkginfo extension: .plist, .pkginfo and no extensions are the most commonly used options, all equally acceptable

editor: What is your preferred text editor?

default catalog: **show most interesting guy in the world**

configure munkiimport

munkiimport is a command line tool (it's one of the Python scripts) for importing software into our munki repo. It was installed with the munki tools, and now we need to tell it how our munki repository is configured, and what tools we prefer to edit text with. *walk through each line*

Path: where is that website you're serving?

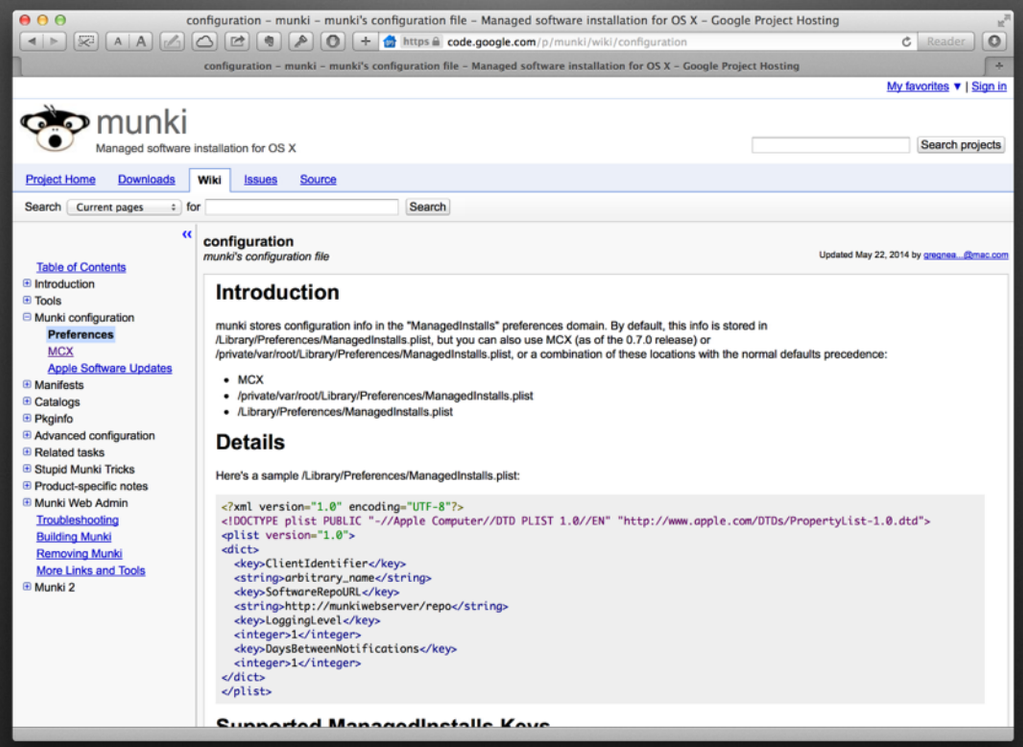
Repo fileshare: relevant if you're importing to a remote machine. We're not.

pkginfo extension: .plist, .pkginfo and no extensions are the most commonly used options, all equally acceptable

editor: What is your preferred text editor?

default catalog: **show most interesting guy in the world**

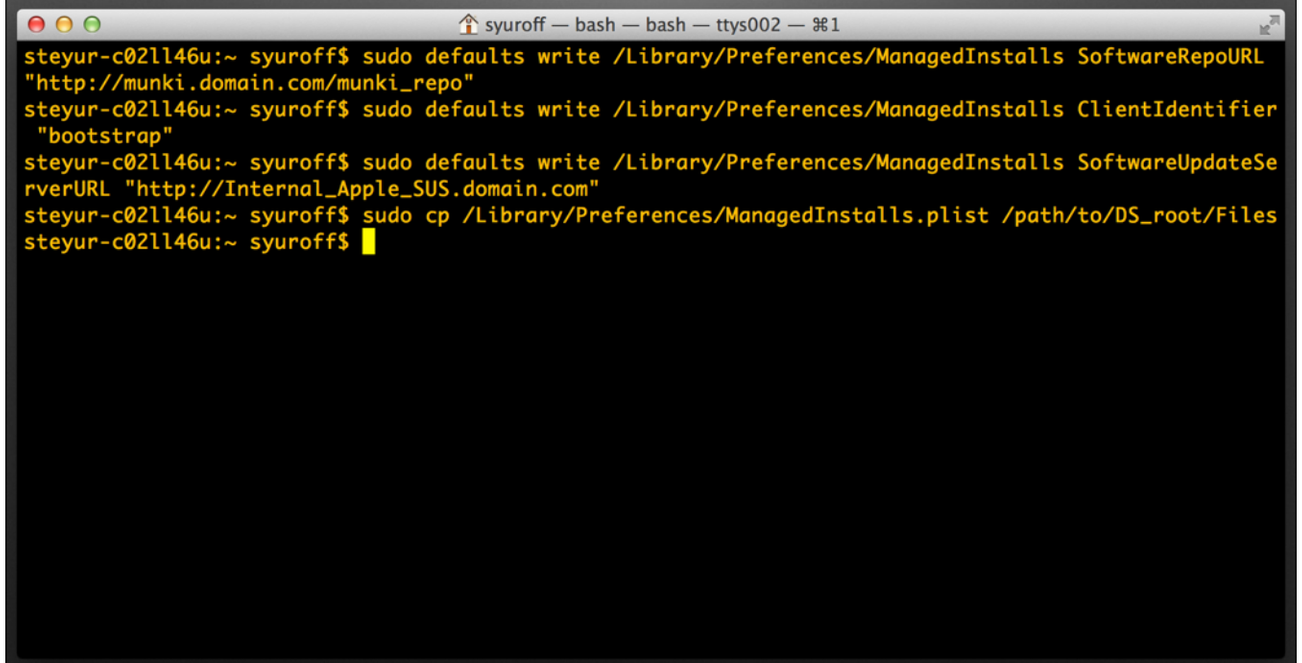
default managedinstalls.plist



Besides having munki tools installed, each client needs to know what munki server to talk to and what manifest to request. That configuration will be saved in /Library/Preferences/ManagedInstalls.plist

There are multiple ways to get this starting file: Download from google code or git clone, or use **defaults write** to create your own.

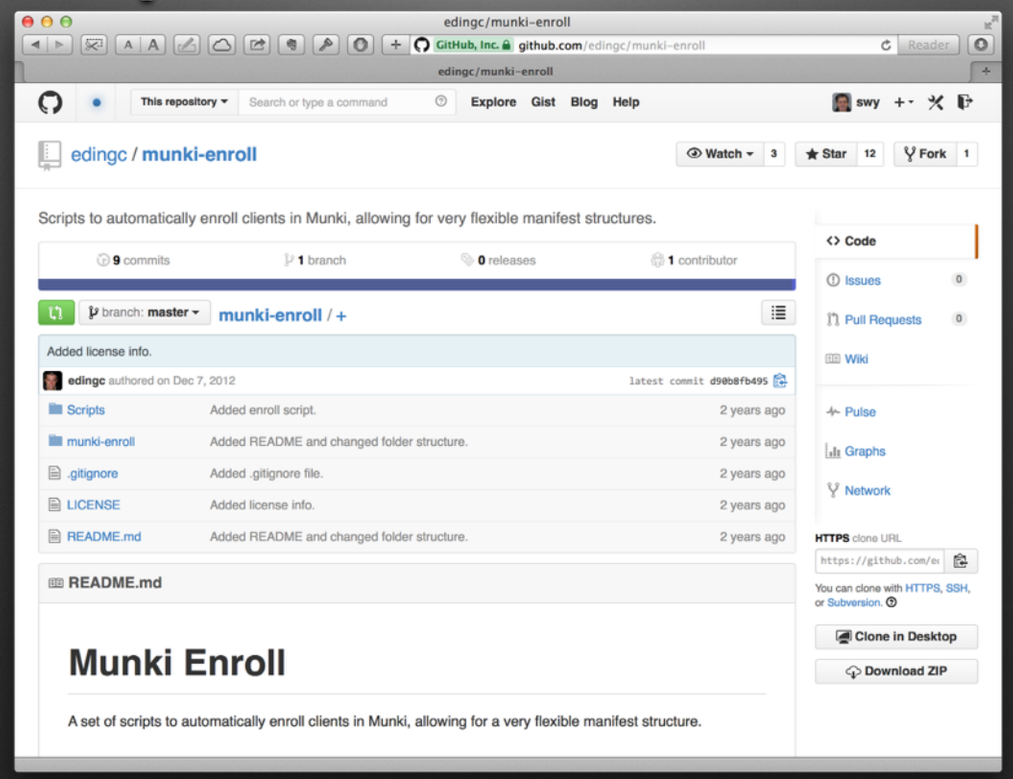
default managedinstalls.plist



Besides having munki tools installed, each client needs to know what munki server to talk to and what manifest to request. That configuration will be saved in /Library/Preferences/ManagedInstalls.plist

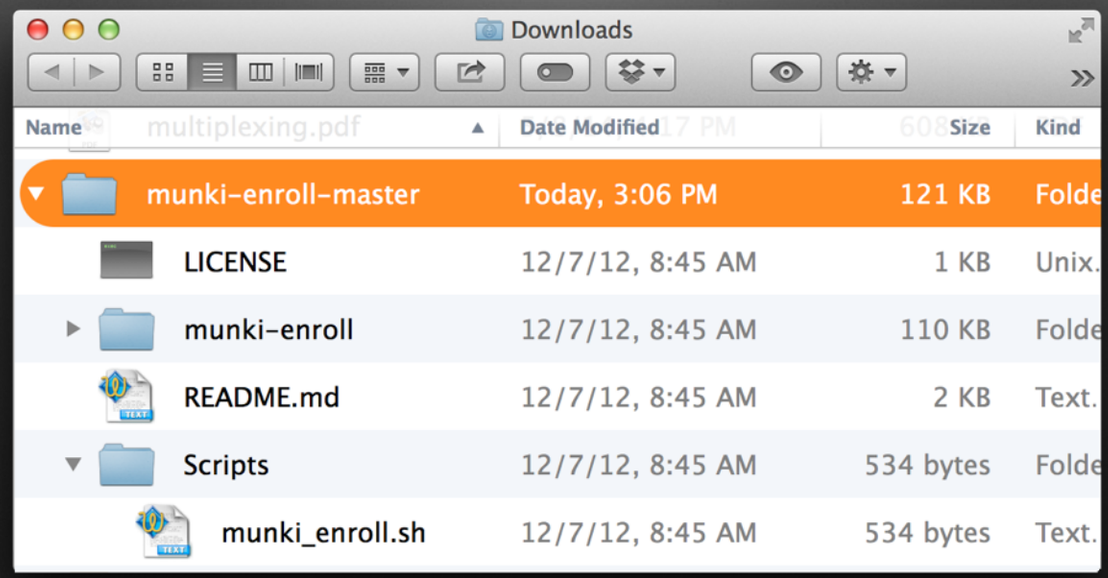
There are multiple ways to get this starting file: Download from google code or git clone, or use **defaults write** to create your own.

optional: munki-enroll



3rd party add on
Manifest: a file that is stored on the munki server, and indicates what software is to be installed on each client that requests that manifest. It's an order form.
munki-enroll exists to automate making a unique manifest for each machine that checks in to the munki server, and adjusting that machine to seek that new manifest.
You don't have to have unique manifests, but someday you'll probably want it, as eventually you'll have machines with differing software on each. This handles the work for you. Manifests can include other manifests: so you may end up with many manifests that don't install anything on their own, but just call the main manifest, but your framework is set for the day one needs to be different from the default.
How do we use it? You can just grab the .zip file, and expand

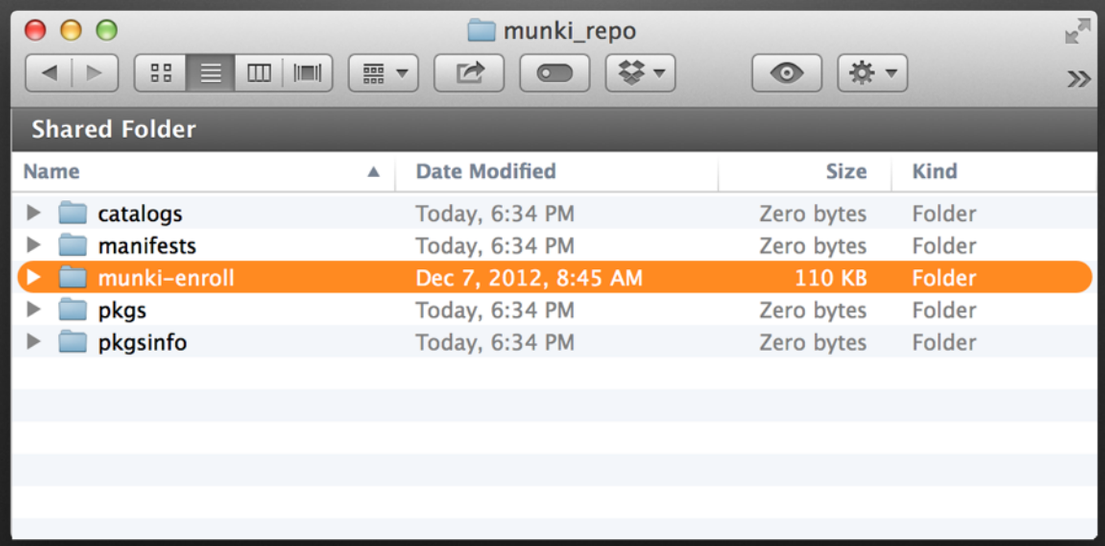
munkienroll: place php



Once expanded, we have 3 steps: *advance here*

1: Copy the munki-enroll folder to your munki_repo *advance here*

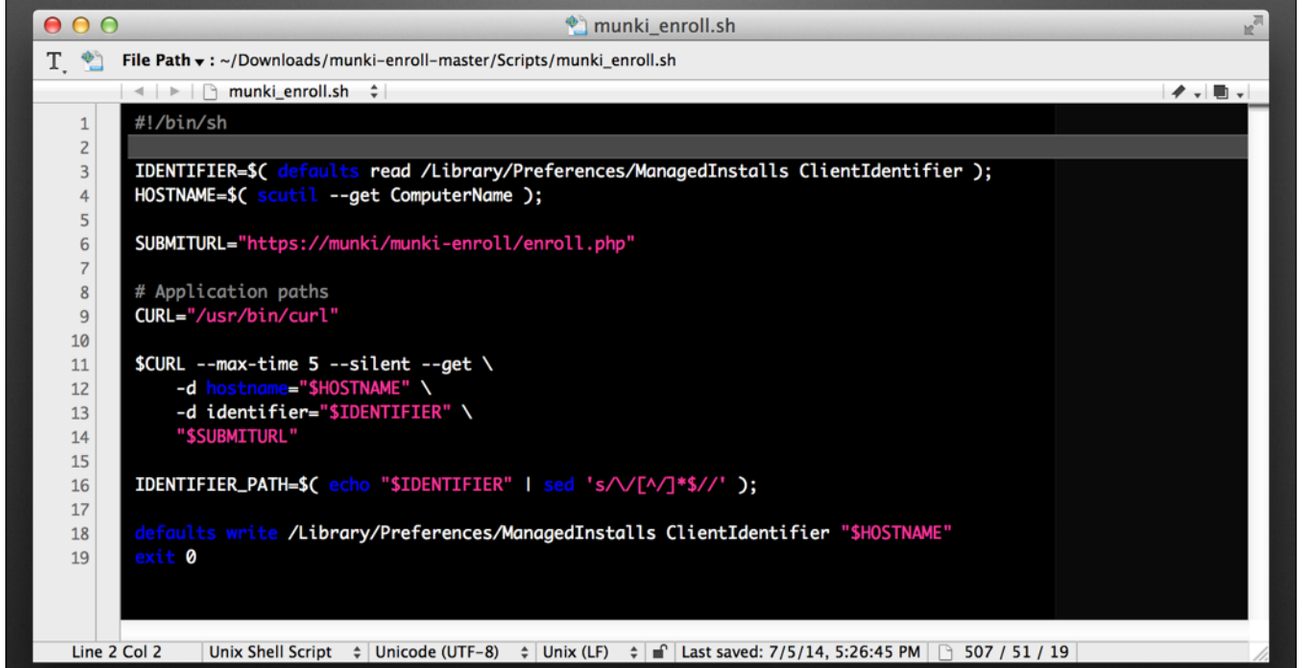
munkienroll: place php



Once expanded, we have 3 steps: *advance here*

1: Copy the munki-enroll folder to your munki_repo *advance here*

munkienroll: edit script



Step 2: tweak the included munki_enroll.sh script to be appropriate for your setup

Line 6 isn't the right url for your munki server. Fix as appropriate.

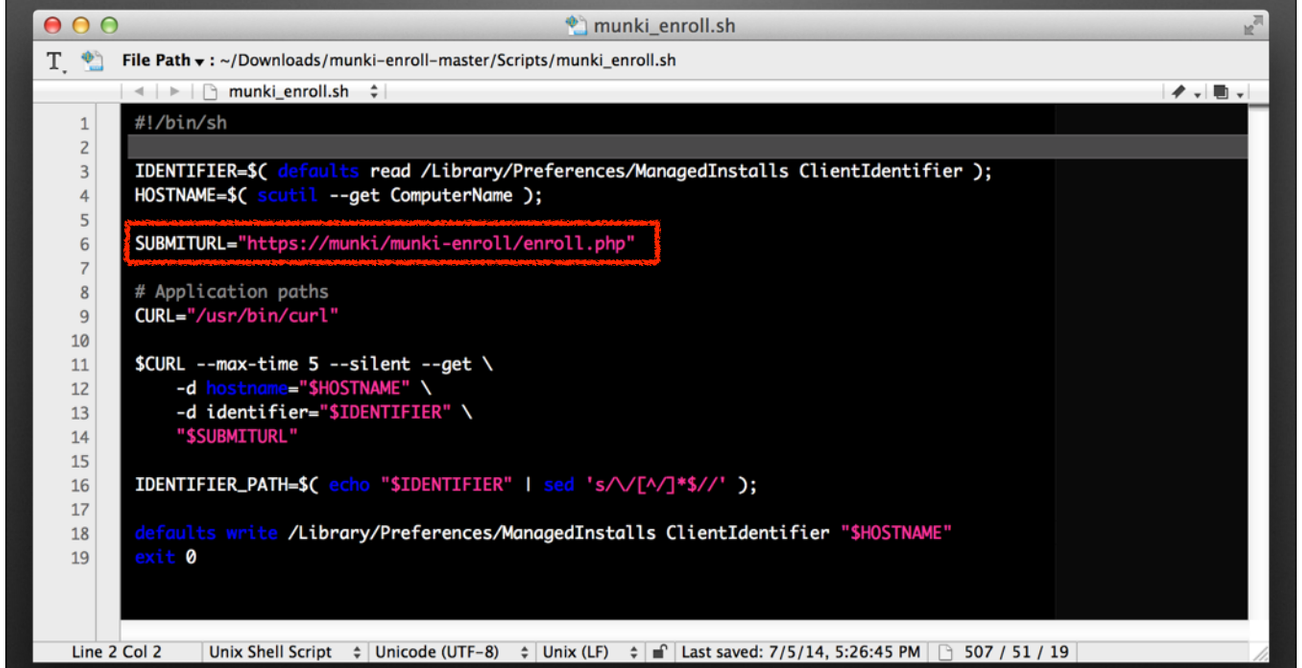
When this script is run by the client, it gathers up the current manifest name and machine name, and by CURLing the PHP script, that PHP makes a new manifest based on machine name, and makes the original manifest an "included manifest" within the new one.

It then updates the client machine to seek that new manifest.

We then need a way to run this script on every machine: there are multiple ways to do this.

Munki supports creating pkginfos with scripts embedded: a type of "nopkg" nopkg has a downside of requiring the admin to provide some sort of tracking of "this script was run" on each client. If a bug or edge case is found, it can get unpleasant to track that the new version was run- you end up reinventing Apple's receipt system.

munkienroll: edit script



Step 2: tweak the included munki_enroll.sh script to be appropriate for your setup

Line 6 isn't the right url for your munki server. Fix as appropriate.

When this script is run by the client, it gathers up the current manifest name and machine name, and by CURLing the PHP script, that PHP makes a new manifest based on machine name, and makes the original manifest an "included manifest" within the new one.

It then updates the client machine to seek that new manifest.

We then need a way to run this script on every machine: there are multiple ways to do this.

Munki supports creating pkginfos with scripts embedded: a type of "nopkg" nopkg has a downside of requiring the admin to provide some sort of tracking of "this script was run" on each client. If a bug or edge case is found, it can get unpleasant to track that the new version was run- you end up reinventing Apple's receipt system.

package munkienroll script

To make a package that runs a script and leaves a receipt, we'll use Greg Neagle's Payload Free package template, from managingOSX.wordpress.com. With this, Greg has provided a template pkg and documentation of which values in which files need to be edited. To access the files, right click on the package, and expand package contents

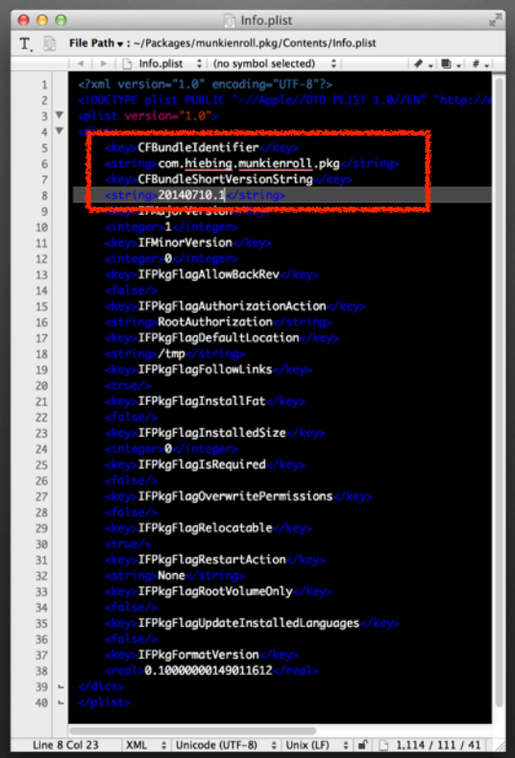
2) Contents/Resources/en.lproj/description.plist: edit description and title as you see fit

3) edit the postflight file to hold your script

package munkienroll script

1) Contents/info.plist: Edit Identifier and version number

package munkienroll script



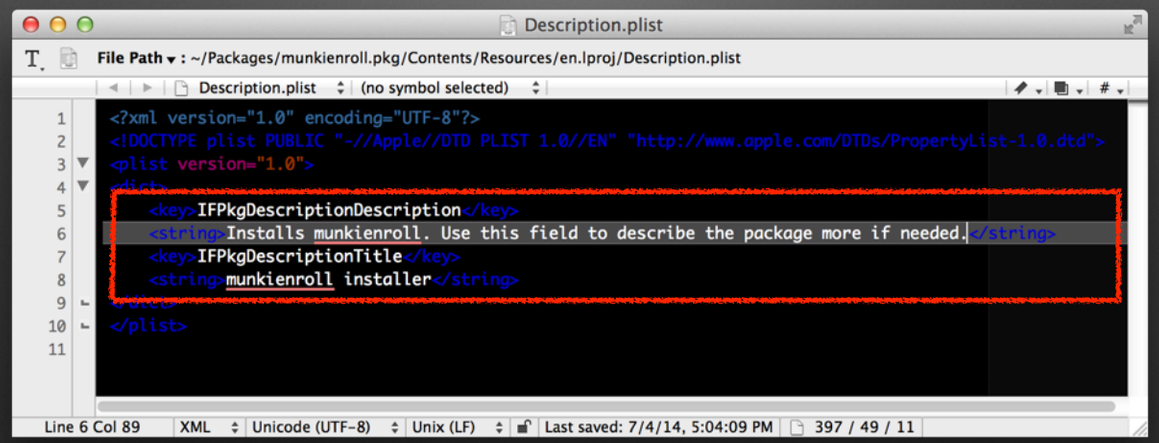
1) Contents/info.plist: Edit Identifier and version number

package munkienroll script

2) Contents/Resources/en.lproj/description.plist: edit description and title as you see fit

3) edit the postflight file to hold your script

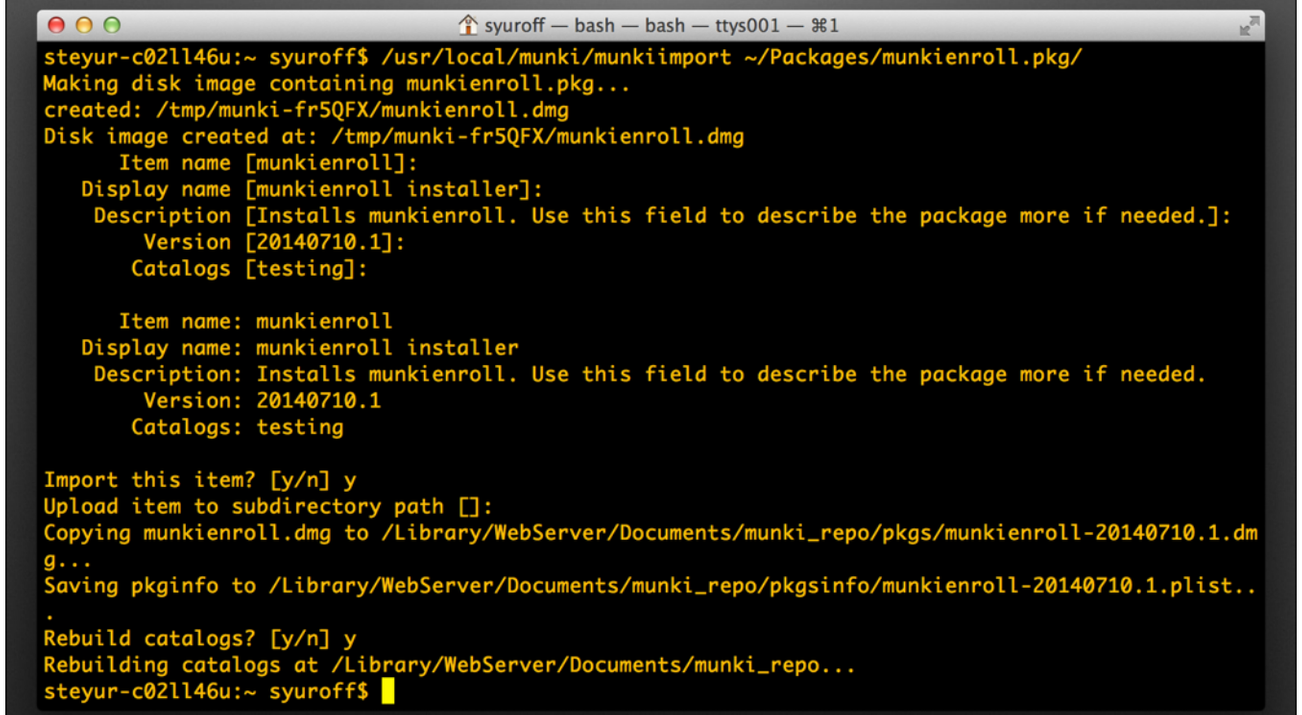
package munkienroll script



2) Contents/Resources/en.lproj/description.plist: edit description and title as you see fit

3) edit the postflight file to hold your script

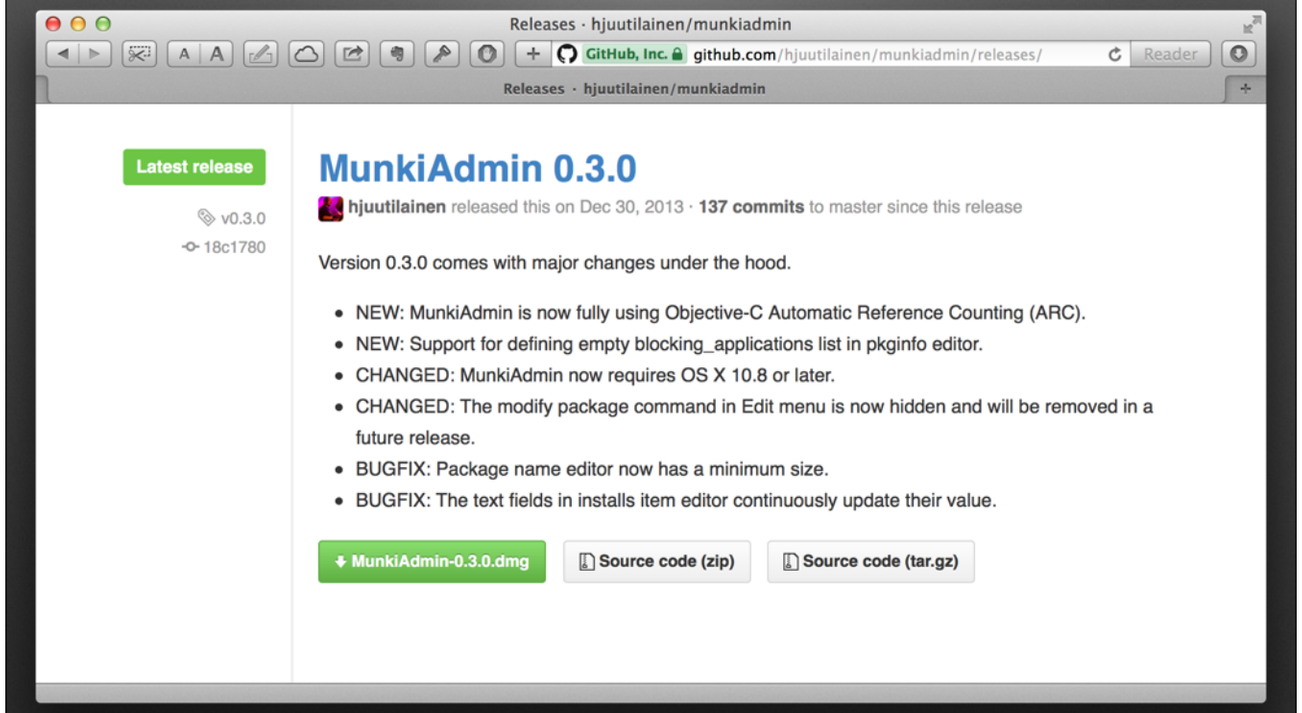
import the munkienroll pkg



Walk through the munkiimport steps here

This is the last command-line interaction with munki we'll do, because I'm going to direct you to...

Get munkiadmin

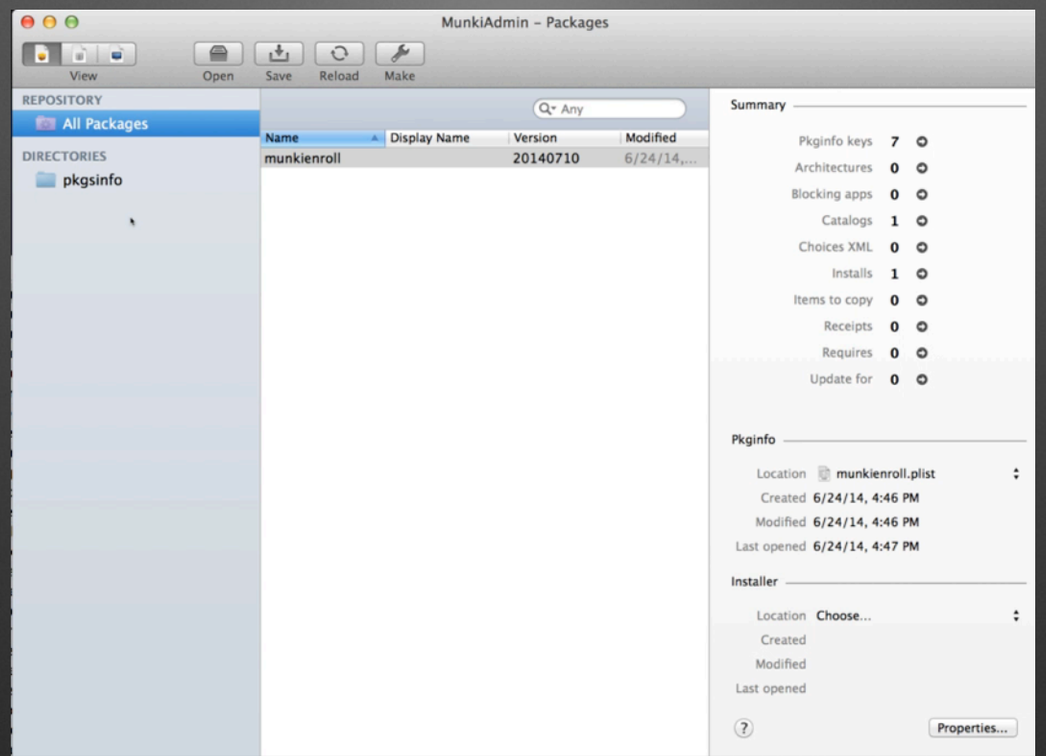


Munkiadmin is a great GUI to manage a munki repository

version 0.3 is stable

Install, on first launch it will ask where your repository is. The answer is your munki_repo folder

Make 2 manifests



THIS IS A MOVIE

In munkiadmin, I make 2 manifests: *hit play*

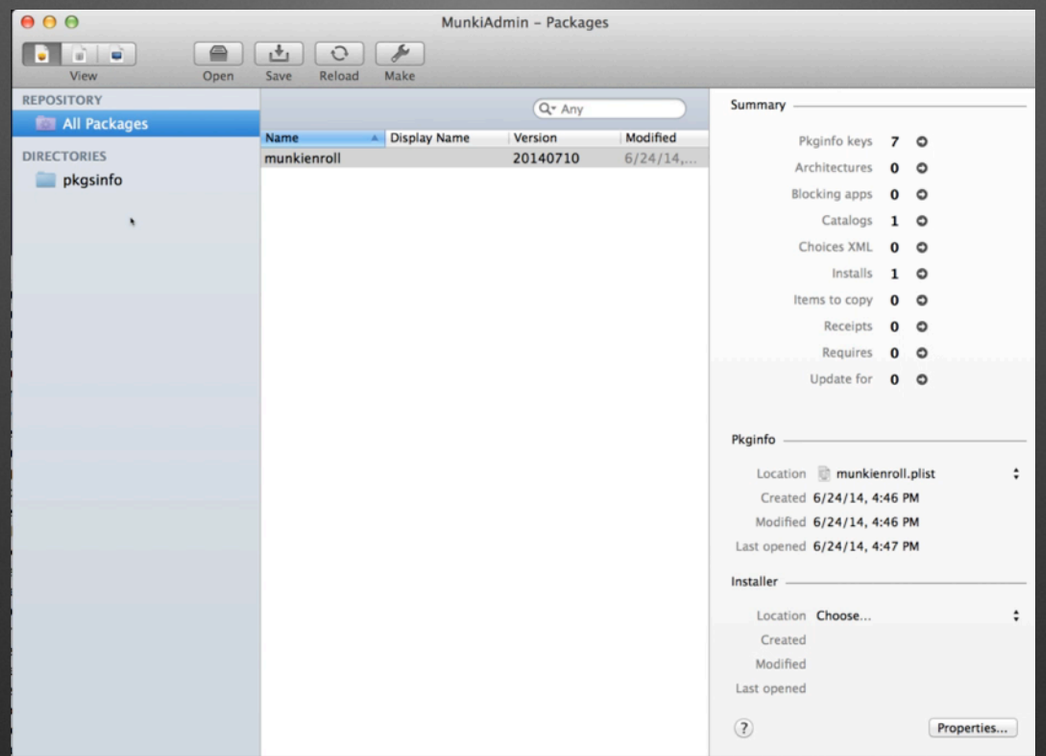
Bootstrap is what each client reads by default. That manifest includes one installation: the munkienroll we just made. It also has one "included manifest" of Universal, which is where I put all approved software.

When a client first connects to the munki server, it reads the default bootstrap manifest, which provides munkienroll pkg, which runs a script, which leads to a new manifest being made and parallel update on the client's ManagedInstalls.plist. Because Universal is an included manifest, this also brings in all the rest of the approved software. From then on, the client will query the manifest made just for it, making it easy to install an application on just one machine, or as many as are needed.

Now we've seen how to use munkiimport to configure munki, and bring in a package manually. Any questions?

Now our munki system needs more software to install. This is where we turn to autopkg

Make 2 manifests



THIS IS A MOVIE

In munkiadmin, I make 2 manifests: *hit play*

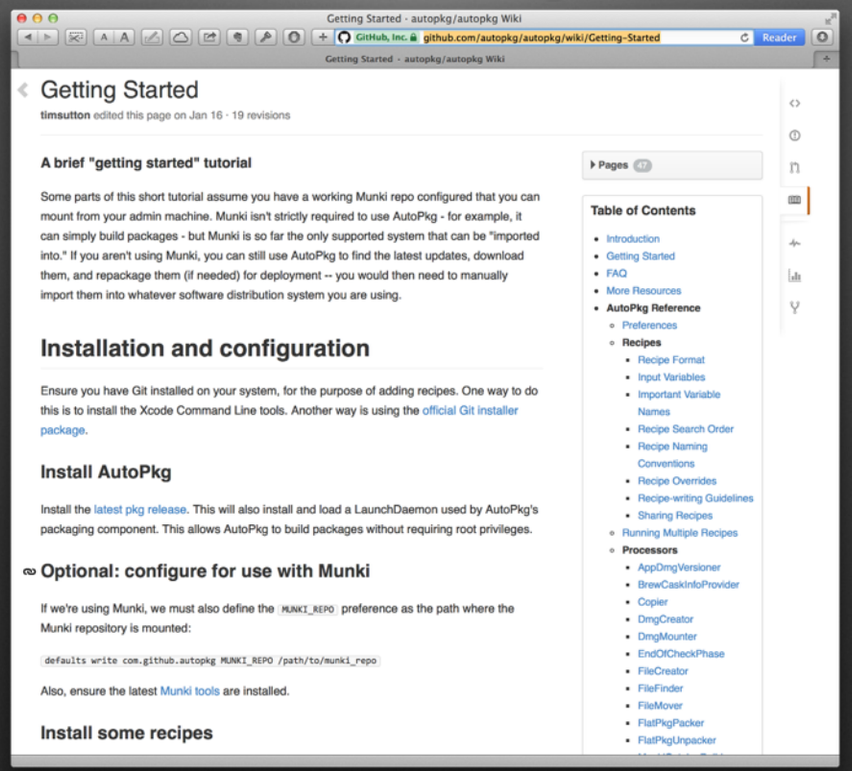
Bootstrap is what each client reads by default. That manifest includes one installation: the munkienroll we just made. It also has one "included manifest" of Universal, which is where I put all approved software.

When a client first connects to the munki server, it reads the default bootstrap manifest, which provides munkienroll pkg, which runs a script, which leads to a new manifest being made and parallel update on the client's ManagedInstalls.plist. Because Universal is an included manifest, this also brings in all the rest of the approved software. From then on, the client will query the manifest made just for it, making it easy to install an application on just one machine, or as many as are needed.

Now we've seen how to use munkiimport to configure munki, and bring in a package manually. Any questions?

Now our munki system needs more software to install. This is where we turn to autopkg

AutoPkg!



Remember: the purpose of autopkg is to simplify downloading and preparing freely distributed software for installation, and making sure you have the most recent version. It can be used as a standalone tool to download, and adjust packages as needed, but playing hand in hand with Munki was a design consideration, and it does it well.

On the server, obtain and run installer from its page on github.

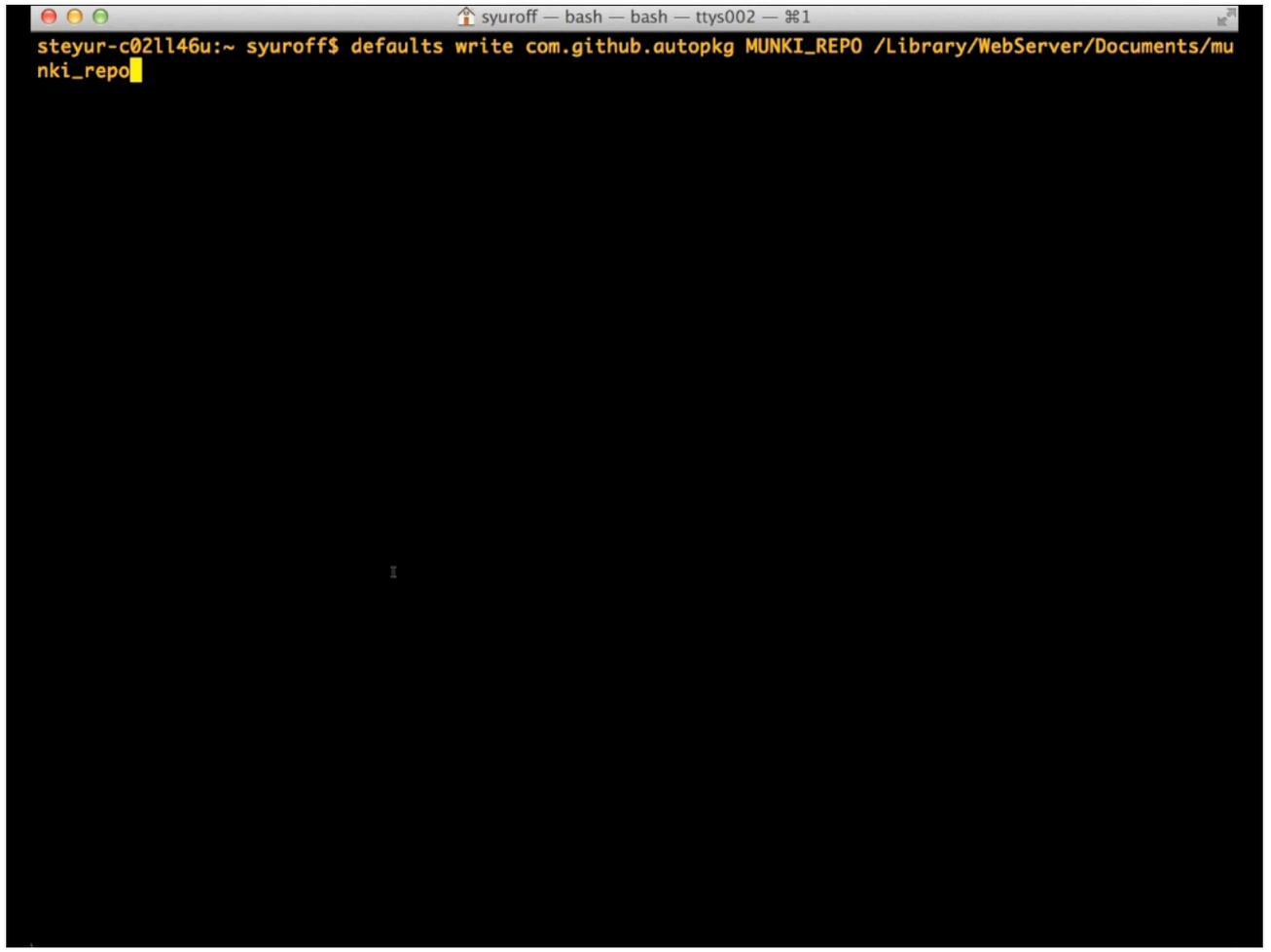
AutoPkg! Now what?

- Server must have git installed
  - via Xcode command line tools
  - or http://git-scm.com/download/mac

Before we go any further: autopkg server needs git.

What is git? It's a version control system: allows us to store software from multiple authors in a public place and control versions. Public Git repos are key to expanding autopkg's vocabulary.

Worried that you don't speak git? Don't be: you don't have to know anything about git to use AutoPkg.



This is a movie

Once autopkg is installed, we have to configure it. We do that by opening Terminal, and teaching it where our munki repo is, because we need the software it downloads to end up in the repo.

So if we simply run the command "autopkg", we see what commands autopkg understands. A key concept in autopkg is **recipes**, as recipes are what do all of the work. When we list recipes, we see that a fresh autopkg install has exactly 0 of them.

We need to get some recipes, via the repo-add command. Adding "recipies" will bring you the default group.

Now when we list recipes, the results are very different. Autopkg says "I know how to get all these things!"

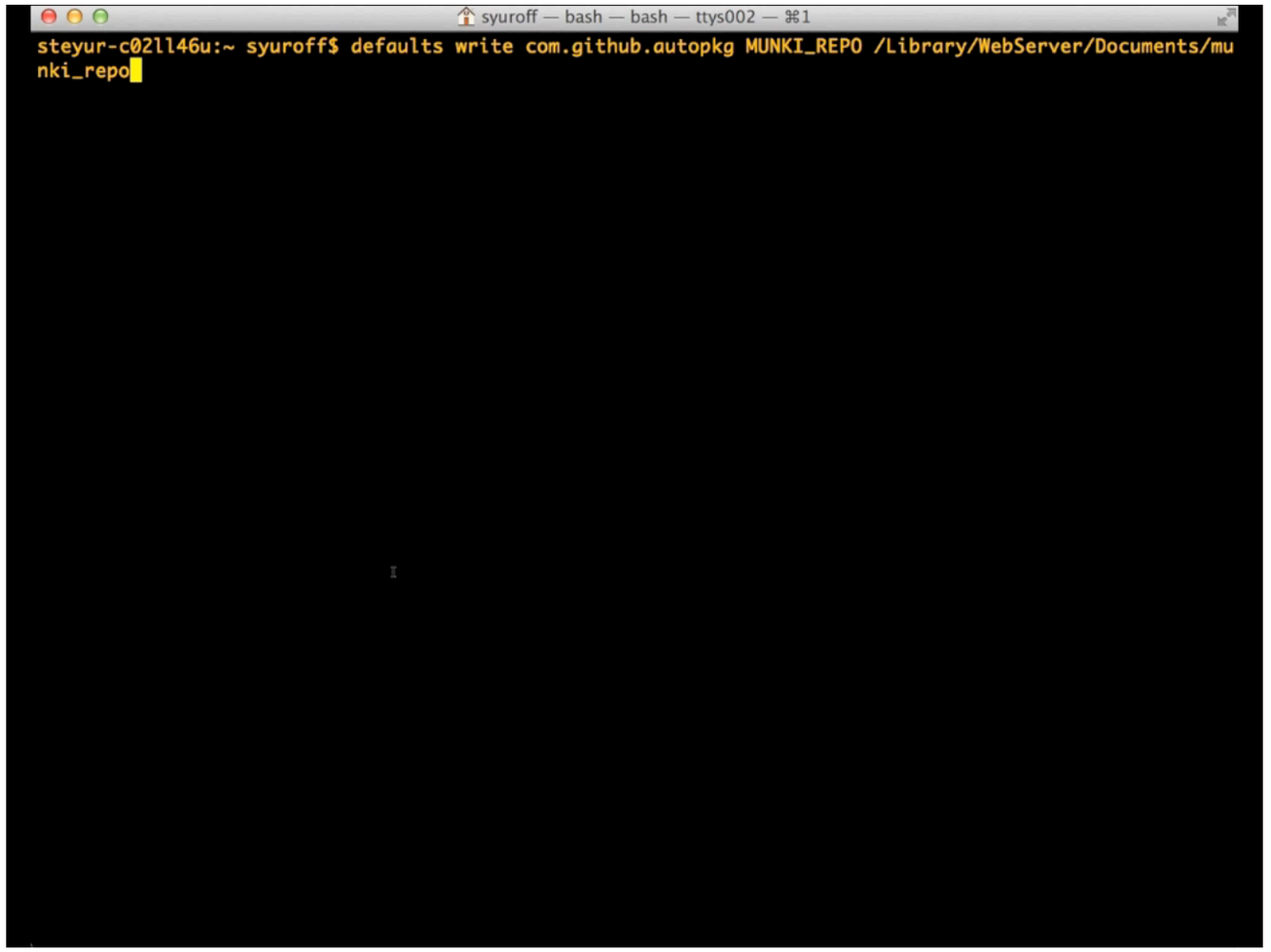
Great, autopkg. Please get us the current release of Chrome and Adobe Air.

You'll see that Chrome is a very simple download.

Adobe air, however, gets a lot more processing. That's because the Air installer as Adobe provides it, won't install silently, in the background But with autopkg, that's not a problem because the recipe fixes it, thanks to some fellow macadmins.

If I flip over to munkiadmin and refresh, we see munki has 2 new offerings in its inventory.

So lets say I need to deploy another app, named Fuze. First thing I'd do is *autopkg search* for it. I see it's not in a repo I already have, so I can *repo-add* that repo, and autopkg run fuze.munki.



This is a movie

Once autopkg is installed, we have to configure it. We do that by opening Terminal, and teaching it where our munki repo is, because we need the software it downloads to end up in the repo.

So if we simply run the command "autopkg", we see what commands autopkg understands. A key concept in autopkg is **recipes**, as recipes are what do all of the work. When we list recipes, we see that a fresh autopkg install has exactly 0 of them.

We need to get some recipes, via the repo-add command. Adding "recipies" will bring you the default group.

Now when we list recipes, the results are very different. Autopkg says "I know how to get all these things!"

Great, autopkg. Please get us the current release of Chrome and Adobe Air.

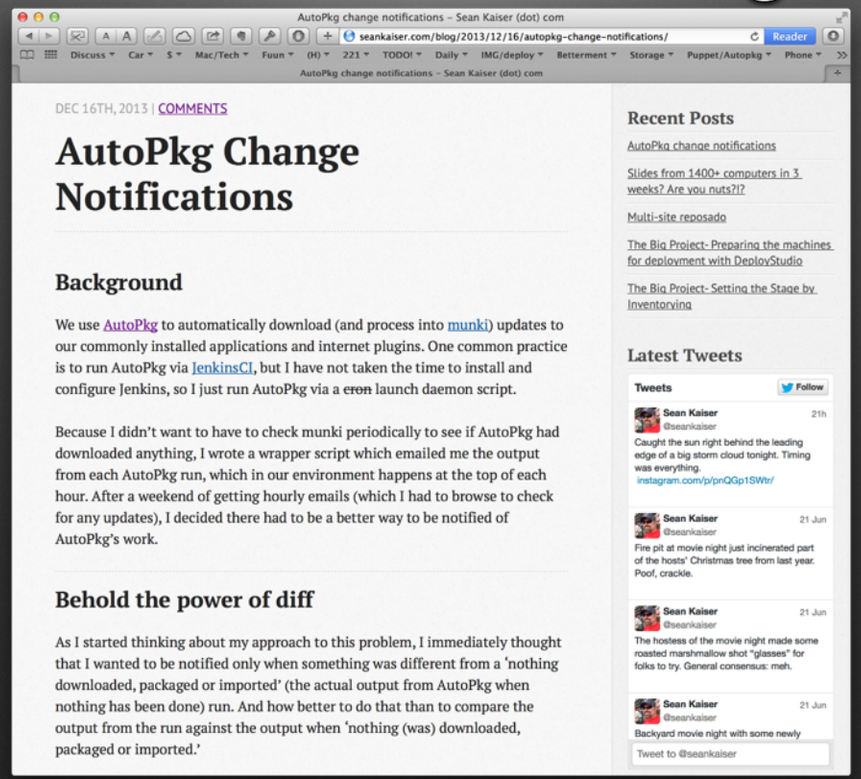
You'll see that Chrome is a very simple download.

Adobe air, however, gets a lot more processing. That's because the Air installer as Adobe provides it, won't install silently, in the background But with autopkg, that's not a problem because the recipe fixes it, thanks to some fellow macadmins.

If I flip over to munkiadmin and refresh, we see munki has 2 new offerings in its inventory.

So lets say I need to deploy another app, named Fuze. First thing I'd do is *autopkg search* for it. I see it's not in a repo I already have, so I can *repo-add* that repo, and autopkg run fuze.munki.

automate AutoPkg



It's awesome that community-contributed recipes can help AutoPkg grab software and import it to your munki repository, and ensure it's kept current, but even better is when it happens while you do other things, then sends you an email IFF something happened.

Sean Kaiser has provided a handy script and accompanying launchdaemon to configure appropriately.

Setup is simple: edit 3 lines that define

automate AutoPkg

The recipes to run *advance*

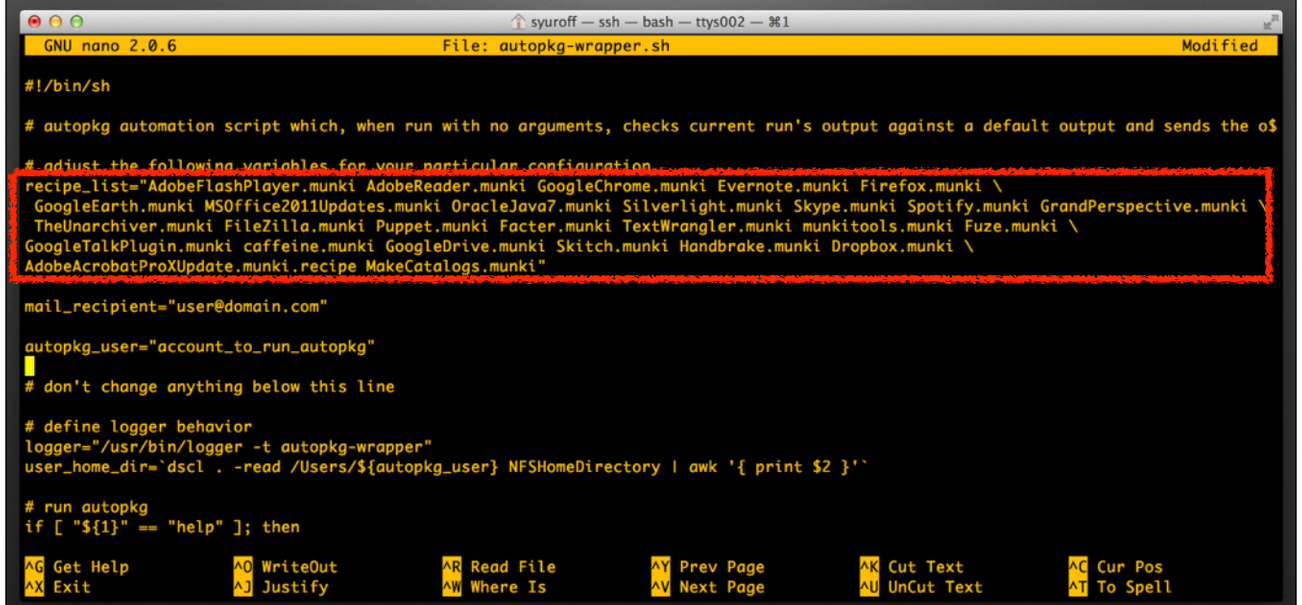
The email address to send alerts to *advance*

The admin account to run the autopkg commands

Run script once manually with an *initialize* option to set the baseline and write a file to compare to, then subsequent runs report if output is different than the base, which indicates a software update.

Schedule this, and your munki repo will always have the latest versions in the testing catalog, and will email you when something was imported.

automate AutoPkg



The recipes to run *advance*

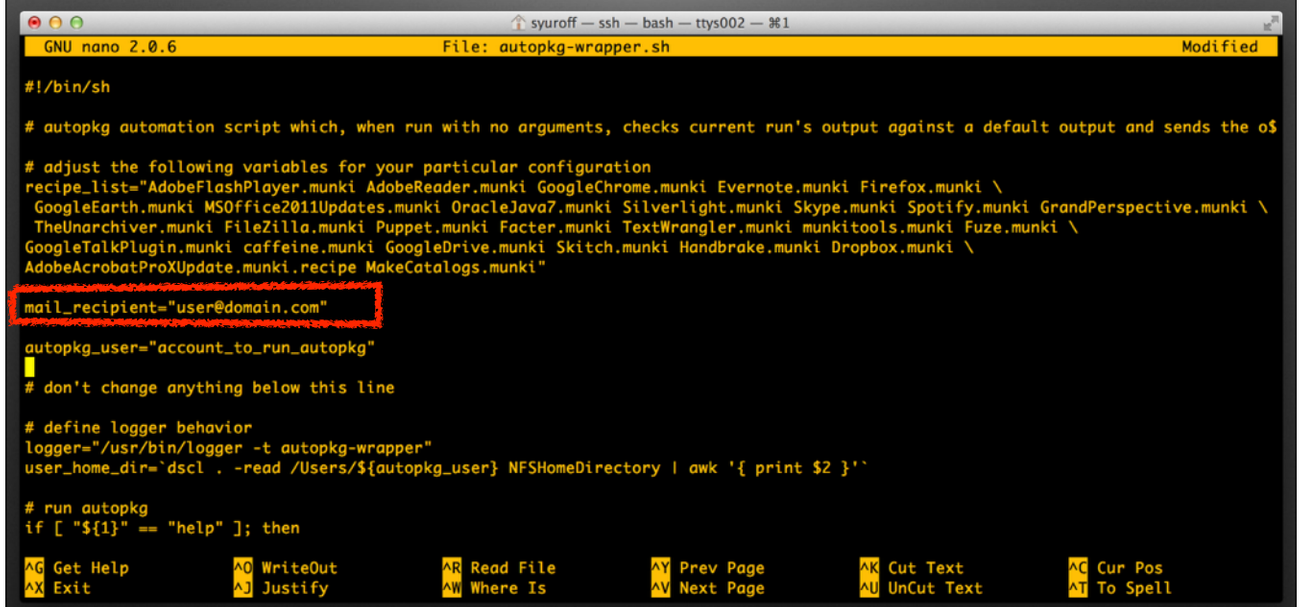
The email address to send alerts to *advance*

The admin account to run the autopkg commands

Run script once manually with an *initialize* option to set the baseline and write a file to compare to, then subsequent runs report if output is different than the base, which indicates a software update.

Schedule this, and your munki repo will always have the latest versions in the testing catalog, and will email you when something was imported.

automate AutoPkg



The recipes to run *advance*

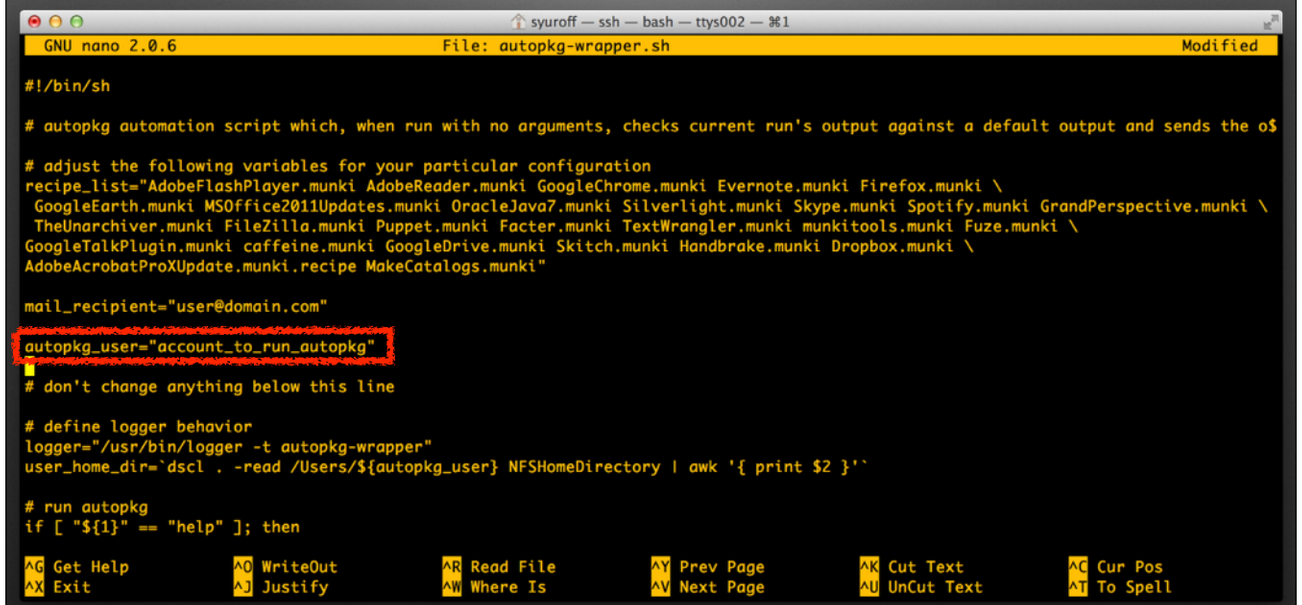
The email address to send alerts to *advance*

The admin account to run the autopkg commands

Run script once manually with an *initialize* option to set the baseline and write a file to compare to, then subsequent runs report if output is different than the base, which indicates a software update.

Schedule this, and your munki repo will always have the latest versions in the testing catalog, and will email you when something was imported.

automate AutoPkg



The recipes to run *advance*

The email address to send alerts to *advance*

The admin account to run the autopkg commands

Run script once manually with an *initialize* option to set the baseline and write a file to compare to, then subsequent runs report if output is different than the base, which indicates a software update.

Schedule this, and your munki repo will always have the latest versions in the testing catalog, and will email you when something was imported.

automate AutoPkg

repo.psumac2014 — ssh — bash — ttys002 — ⌘1

Python | ssh

```
 GNU nano 2.0.6          File: com.hiebing.autopkg-wrapper.plist          Modified

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/Prope$
<plist version="1.0">
<dict>
        <key>Label</key>
        <string>com.hiebing.autopkg-wrapper</string>
        <key>ProgramArguments</key>
        <array>
                <string>/usr/local/bin/autopkg-wrapper.sh</string>
        </array>
        <key>StartCalendarInterval</key>
        <array>
                <dict>
                        <key>Hour</key>
                        <integer>8</integer>
                        <key>Minute</key>
                        <integer>0</integer>
                </dict>
                <dict>
                        <key>Hour</key>
                        <integer>14</integer>
                        <key>Minute</key>
                        <integer>0</integer>
                </dict>
        </array>
        <key>UserName</key>
        <string>account_to_run_autopkg</string>
        <key>AbandonProcessGroup</key>

^G Get Help   ^O WriteOut   ^R Read File   ^Y Prev Page   ^K Cut Text   ^C Cur Pos
^X Exit       ^J Justify    ^W Where Is    ^V Next Page   ^U UnCut Text ^T To Spell
```

A launchdaemon is also included to run this script on a schedule.

automate AutoPkg

When output doesn't match the stored version, you'll get an email.

automate AutoPkg

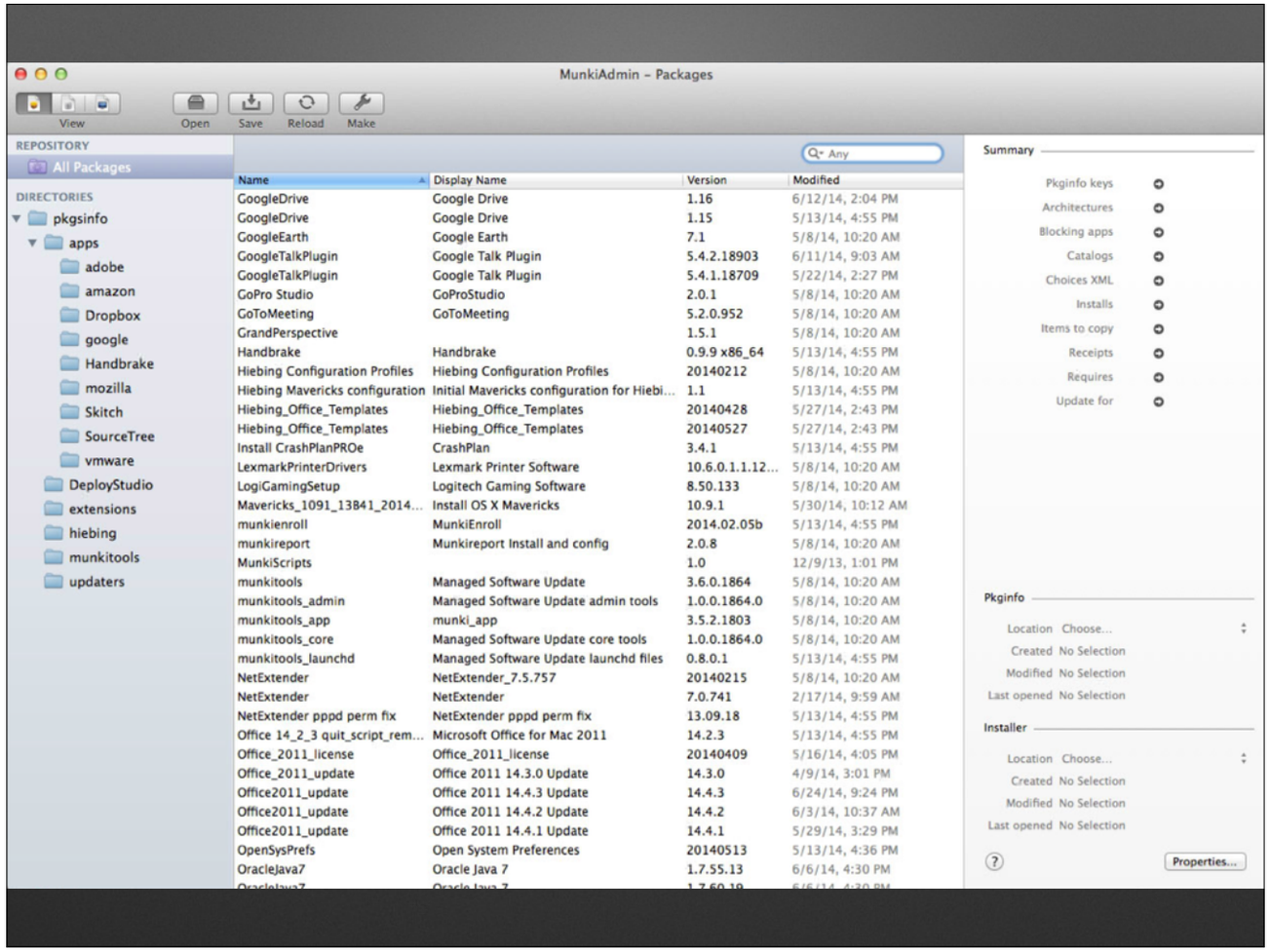
```
Processing AdobeFlashPlayer.munki...
Processing AdobeReader.munki...
Processing GoogleChrome.munki...
Processing Evernote.munki...
Processing Firefox.munki...
Processing GoogleEarth.munki...
Processing MSOffice2011Updates.munki...
Processing OracleJava7.munki...
Processing Silverlight.munki...
Processing Skype.munki...
Processing Spotify.munki...
Processing GrandPerspective.munki...
Processing TheUnarchiver.munki...
Processing FileZilla.munki...
Processing Puppet.munki...
Processing Facter.munki...
Processing TextWrangler.munki...
Processing munkitools.munki...
Processing Fuze.munki...
Processing GoogleTalkPlugin.munki...
Processing caffeine.munki...
Processing GoogleDrive.munki...
Processing Skitch.munki...
Processing Handbrake.munki...
Processing Dropbox.munki...
Processing MakeCatalogs.munki...

The following recipes failed:
    FileZilla.munki
        Error in local.munki.FileZilla: Processor: URLDownloader: Error: Couldn't download http://downloads.sourceforge.net/filezilla/FileZilla_3.8.0_macosx-
x86.app.tar.bz2: HTTP Error 404: Not Found

The following new items were downloaded:
    /Users/mdm01admin/Library/AutoPkg/Cache/local.munki.MSOffice2011Updates/downloads/Office2011-1442Update_EN-US.dmg

The following new items were imported:
    Name                    Version         Catalogs                    Pkginfo Path
    ----                    -------         --------                    ------------
    Office2011_update       14.4.2          testing                     updaters/Office2011_update-14.4.2.plist
```

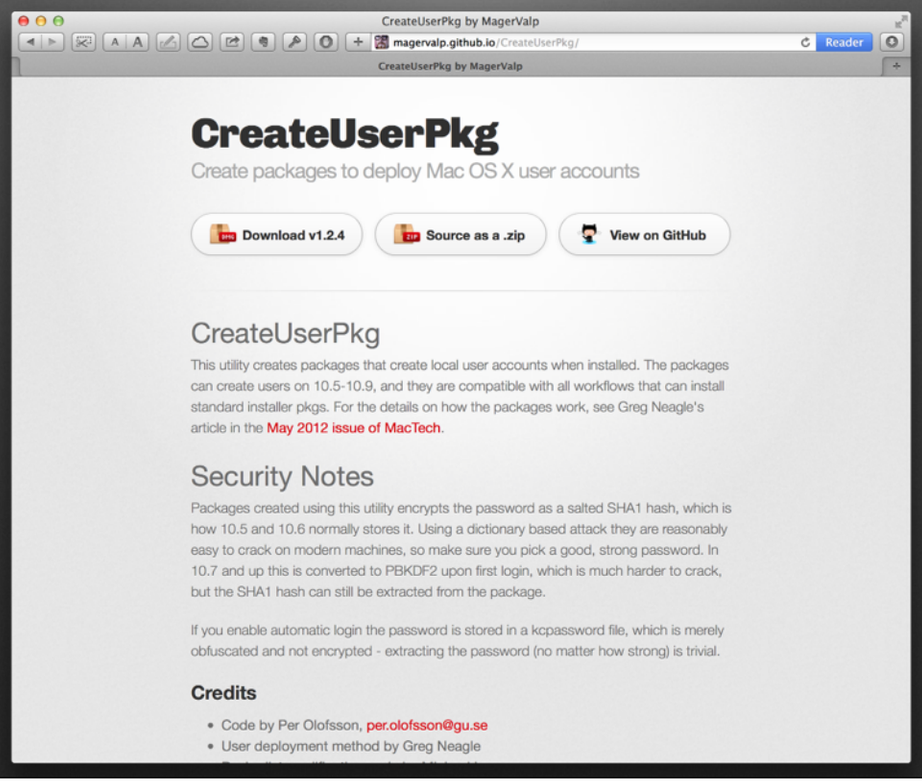
When output doesn't match the stored version, you'll get an email.



Once you've had autopkg up and running for a while, you'll get a very full list of packages in Munkiadmin. You'll probably want to purge these from time to time, as curation does not happen automatically.

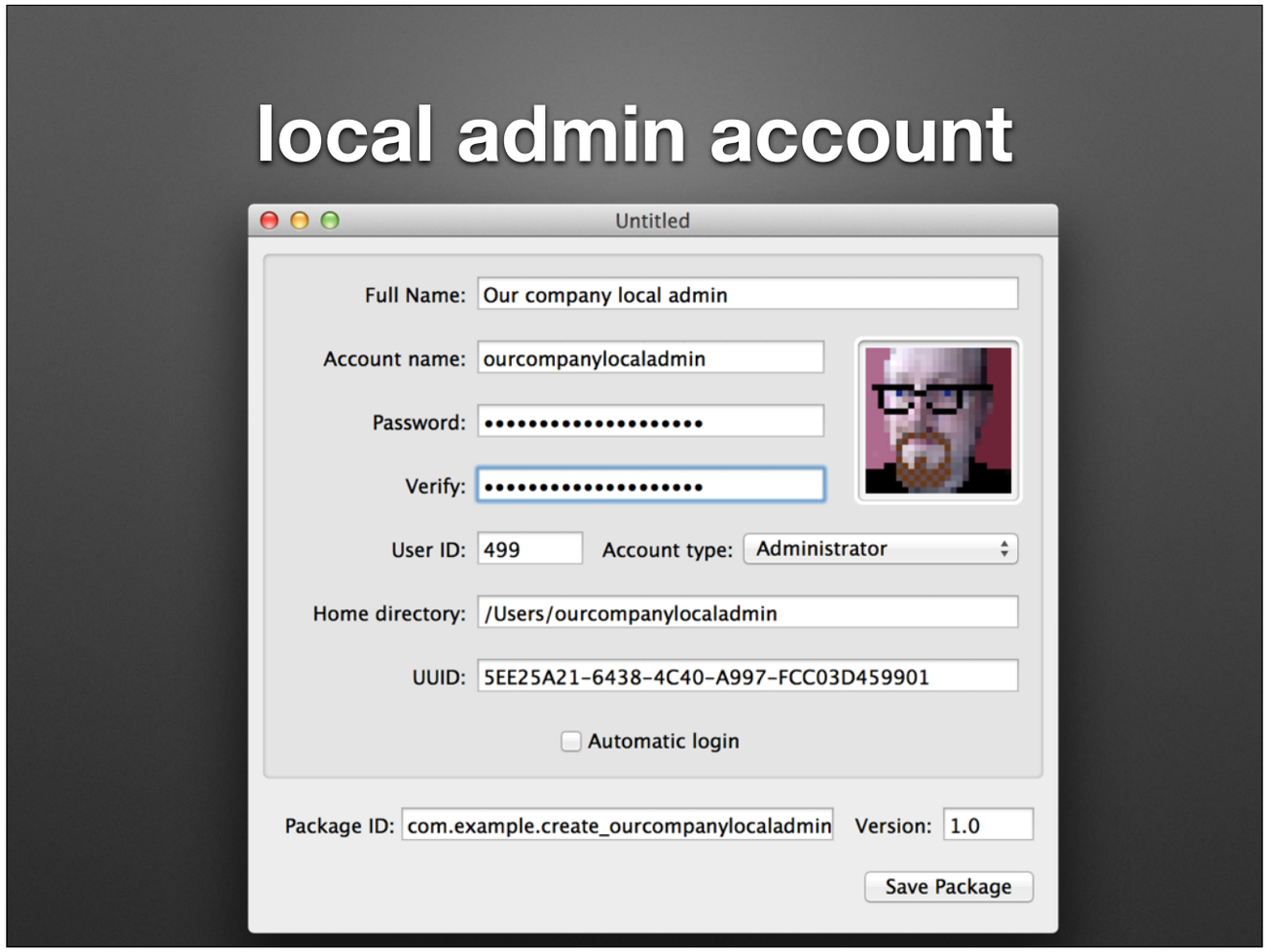
Now we've seen how to install, configure and schedule autopkg. Are there any questions on that section?

local admin account



We've touched on the 4 key elements and munkienroll, so now we'll talk about a few smaller, but essential tools:

Every machine needs a local administrator. Creation of that can be done by a .pkg munkiimported or saved to the proper folder in the deploystudio root, depending on which tool you want to use. Either are acceptable.



We've touched on the 4 key elements and munkienroll, so now we'll talk about a few smaller, but essential tools:

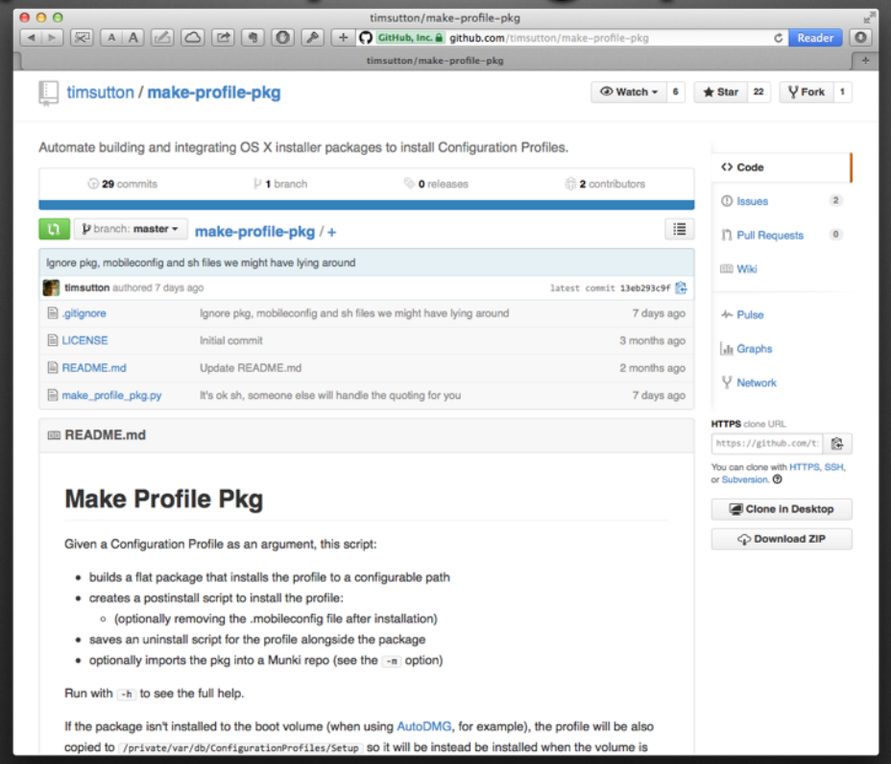
Every machine needs a local administrator. Creation of that can be done by a .pkg munkiimported or saved to the proper folder in the deploystudio root, depending on which tool you want to use. Either are acceptable.

configure firstboot settings

Maybe you don't want every machine to have a local disk named Macintosh HD, or do need to set specific NTP servers, or configure anything else that can be done in a script. Mr Rich Trouton has shared a great starting point for these needs. It's probably very unwise to use this script as-is, directly, but it's a fine starting point.

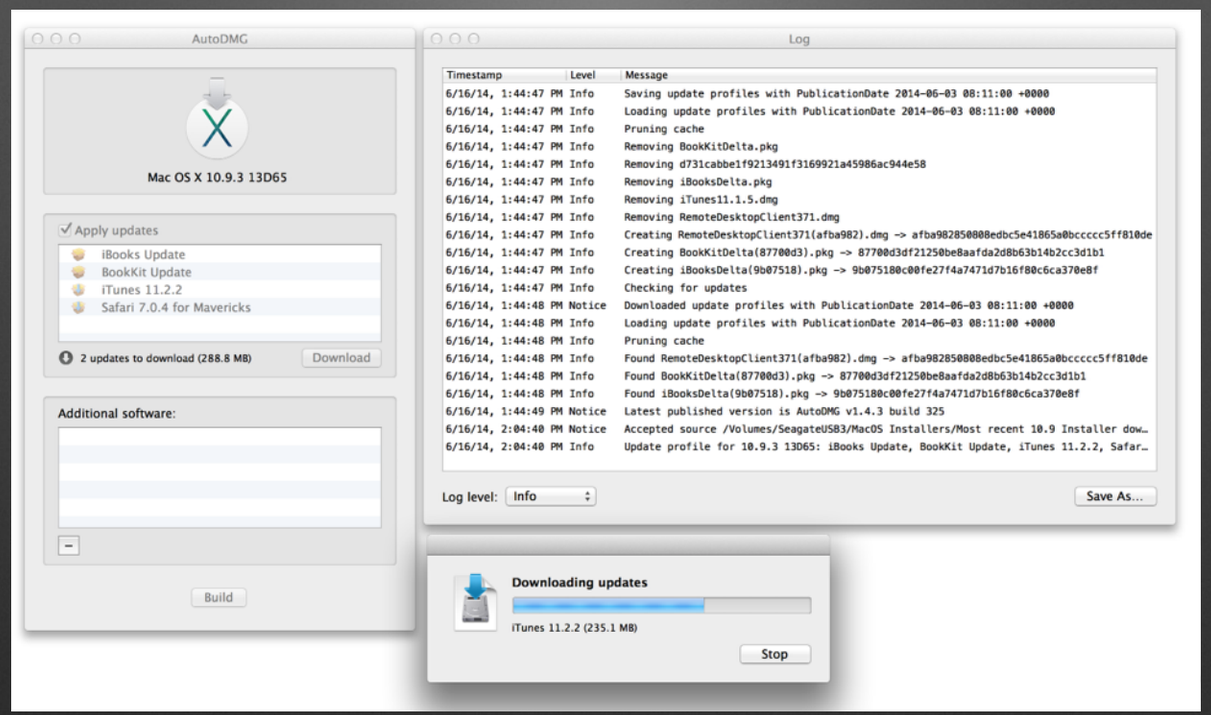
As this is a script that we wish our machines to run, the payload free pkg template we used for the munkienroll package would do nicely to turn your firstboot settings into a pkg, which could be installed via DeployStudio or munki.

optional: package profiles



Profiles are the future for managing settings in OS X. Profiles can be wrapped up in a package, and therefore installed by either munki or DeployStudio. Tim Sutton's make-profile-package is the go-to tool for this purpose.

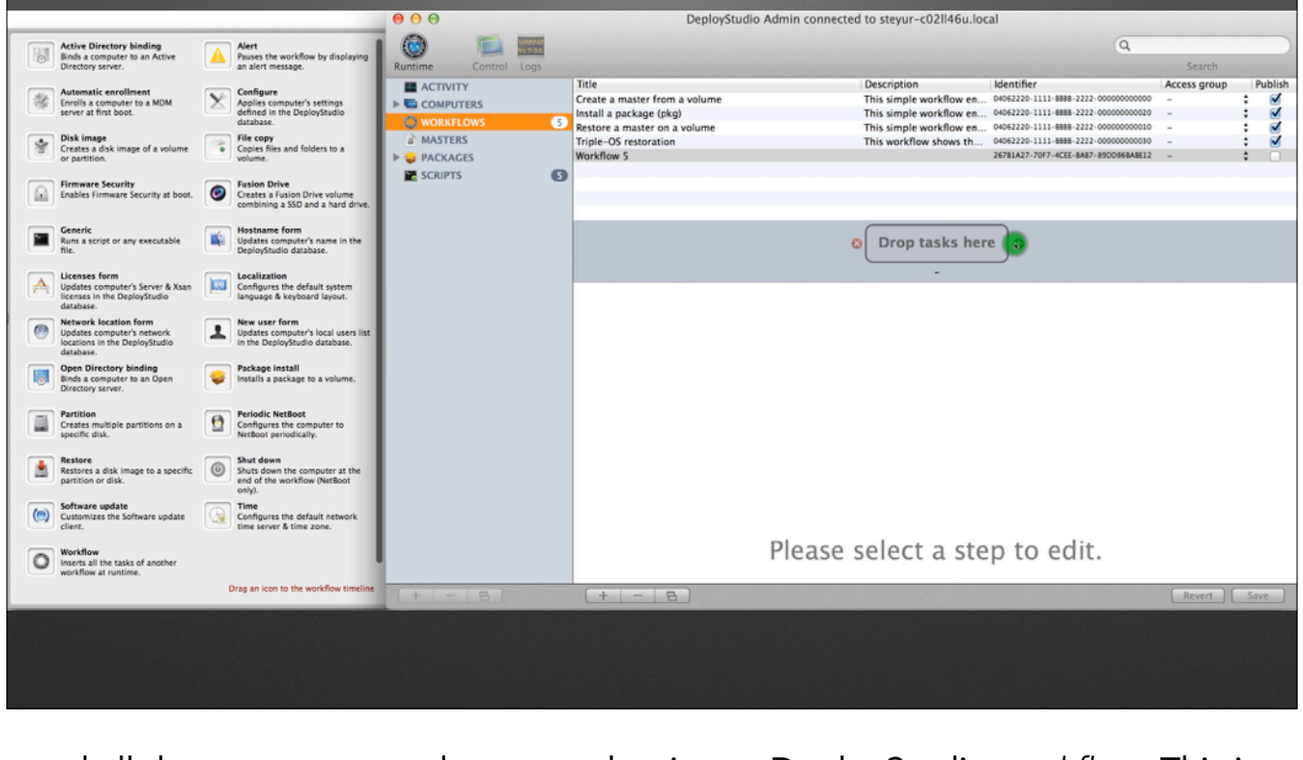
optional: AutoDMG



There can be times where you'll want to completely wipe an existing machine to a default OS install. To generate that never-booted OS, AutoDMG is your tool.

AutoDMG takes an Install OS X app and (optionally) other packages to generate a DMG output, which can be used by DS as a source to restore from.

build a DS workflow



Now that we've organized and configured all the parts, we put them together into a DeployStudio *workflow*. This is made via DeployStudio Assistant

Can have that on the server, or on your admin mac, remotely connected to the DS server

start movie

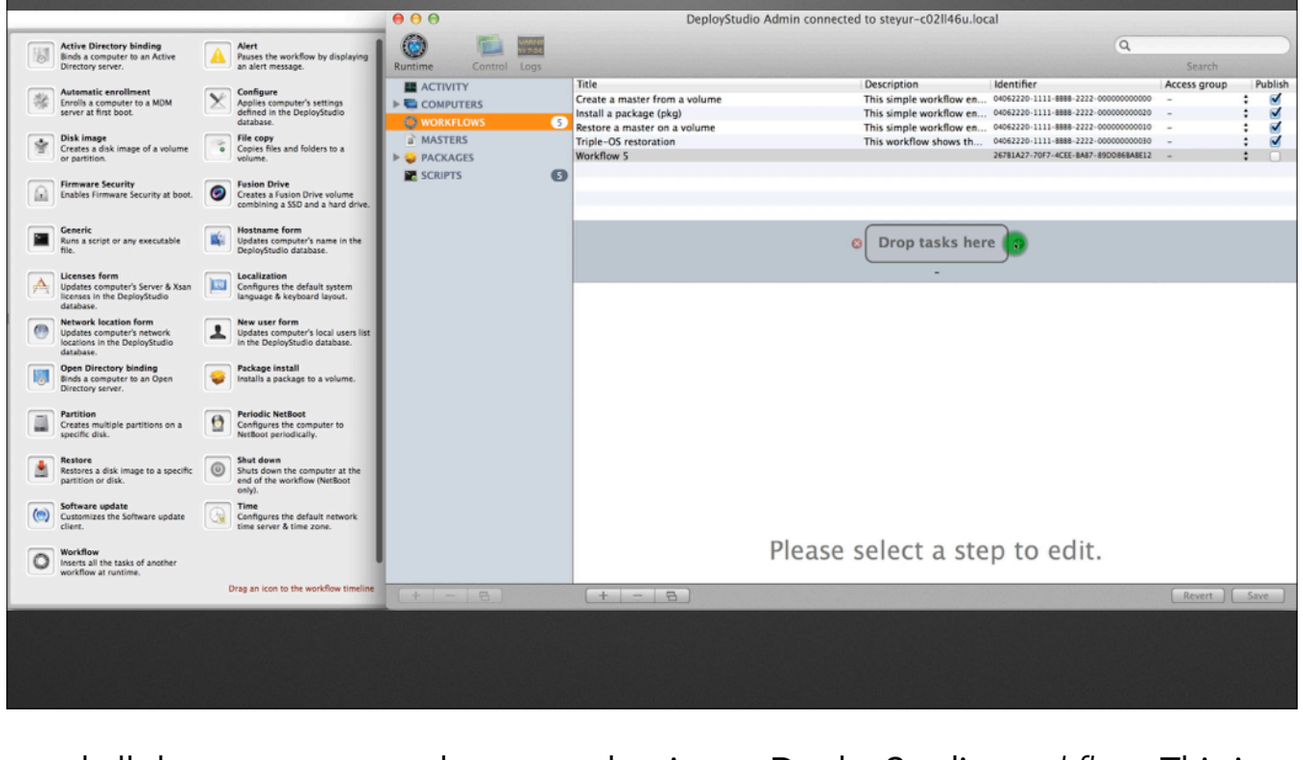
Your newly configured server will look like this: with a few default workflows. We need to make our own, which starts with the [+] at the bottom left.

As you might predict, this makes a blank workflow.

The steps one might take are available via the [+] to the right of "Drop tasks here"

Simply drag them into place- this is how mine is built, but it's not the only way

build a DS workflow



Now that we've organized and configured all the parts, we put them together into a DeployStudio *workflow*. This is made via DeployStudio Assistant

Can have that on the server, or on your admin mac, remotely connected to the DS server

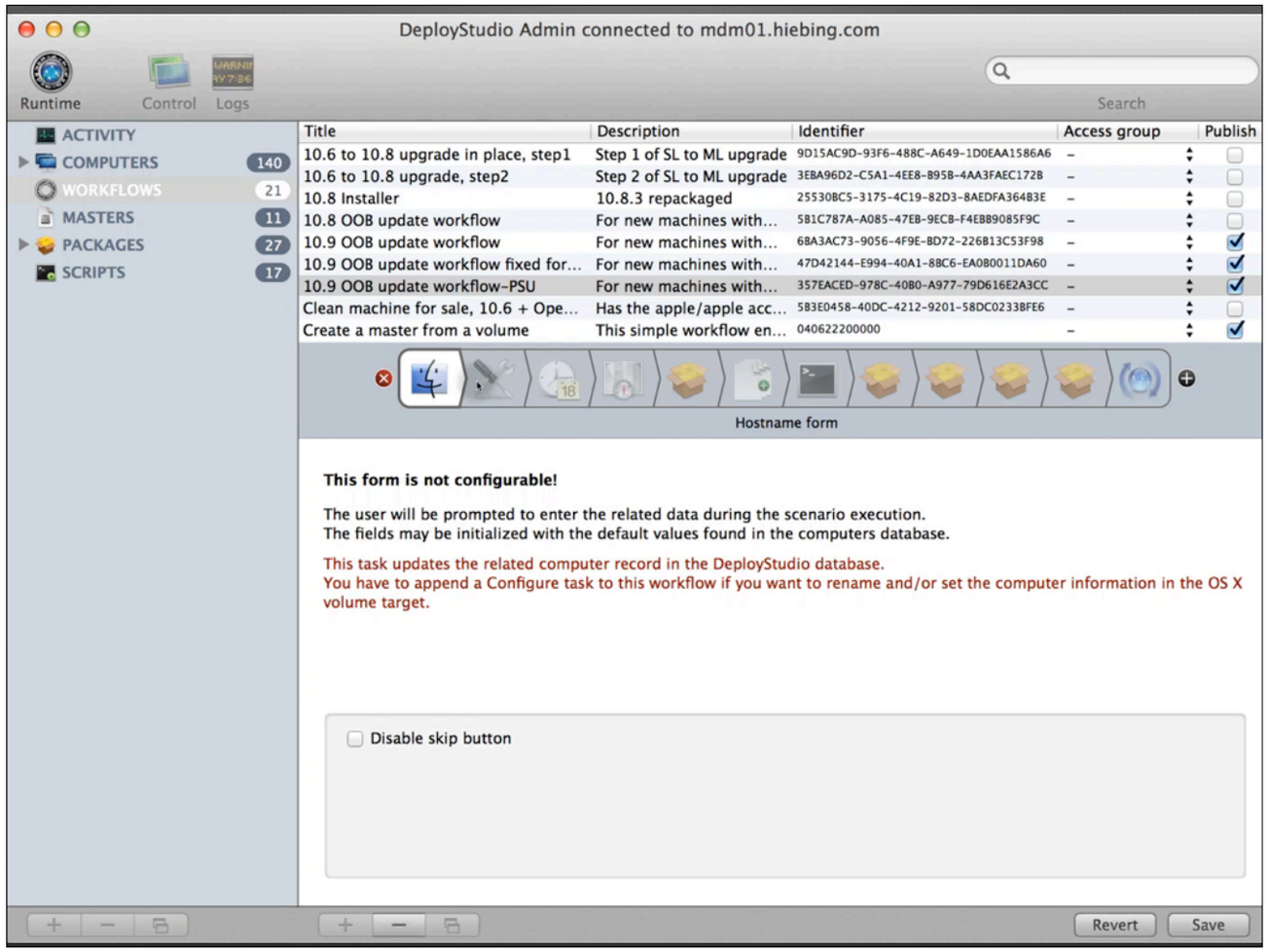
start movie

Your newly configured server will look like this: with a few default workflows. We need to make our own, which starts with the [+] at the bottom left.

As you might predict, this makes a blank workflow.

The steps one might take are available via the [+] to the right of "Drop tasks here"

Simply drag them into place- this is how mine is built, but it's not the only way



When done, my DS workflow looks like this. *this is a movie!*

1) Hostname form: I manually set the name here

2) Configure: this step sets computer options

3) Set the time

4) Bind to AD

5) Create LocalAdmin package

7) Install configuration profiles

8) Munkitools

9) The default Managed Installs.plist

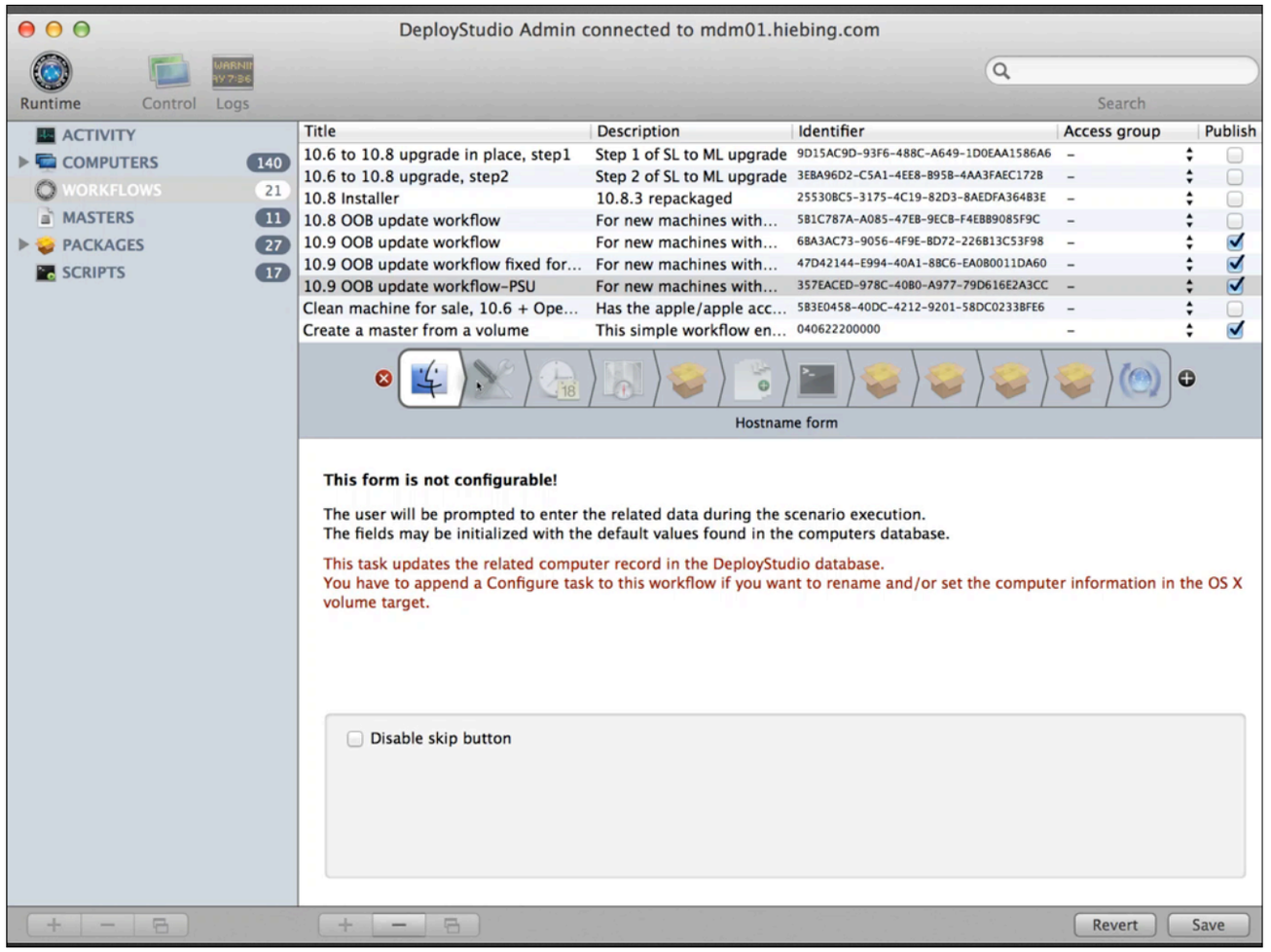
10) ds_touch_munki *see next slide*

11) ManagedInstalls.plist that we made earlier in the session

12) Setting a pkg that sets an on-brand desktop

14) Preconfigure the dock

15) Run software update



When done, my DS workflow looks like this. *this is a movie!*

1) Hostname form: I manually set the name here

2) Configure: this step sets computer options

3) Set the time

4) Bind to AD

5) Create LocalAdmin package

7) Install configuration profiles

8) Munkitools

9) The default Managed Installs.plist

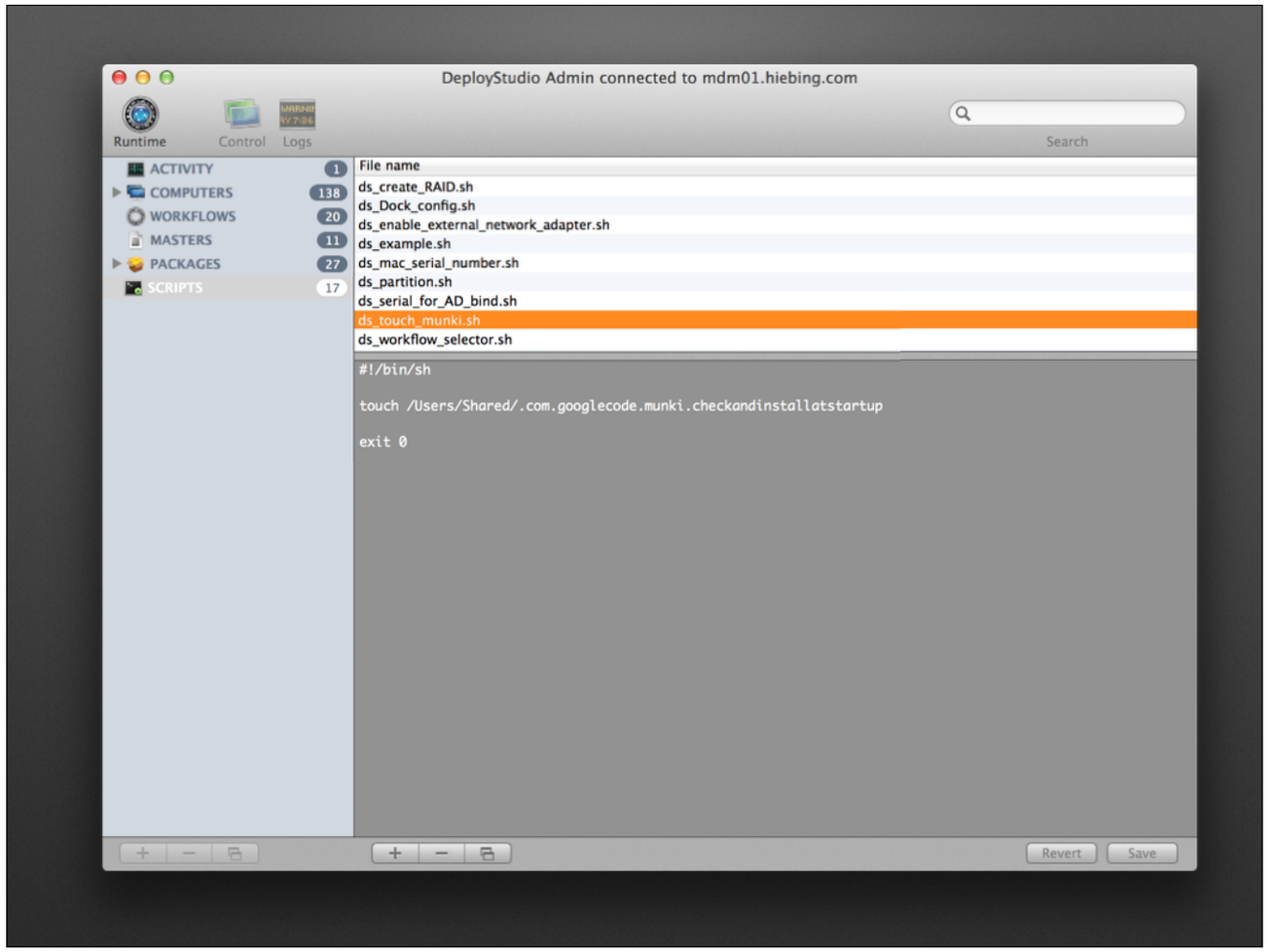
10) ds_touch_munki *see next slide*

11) ManagedInstalls.plist that we made earlier in the session

12) Setting a pkg that sets an on-brand desktop

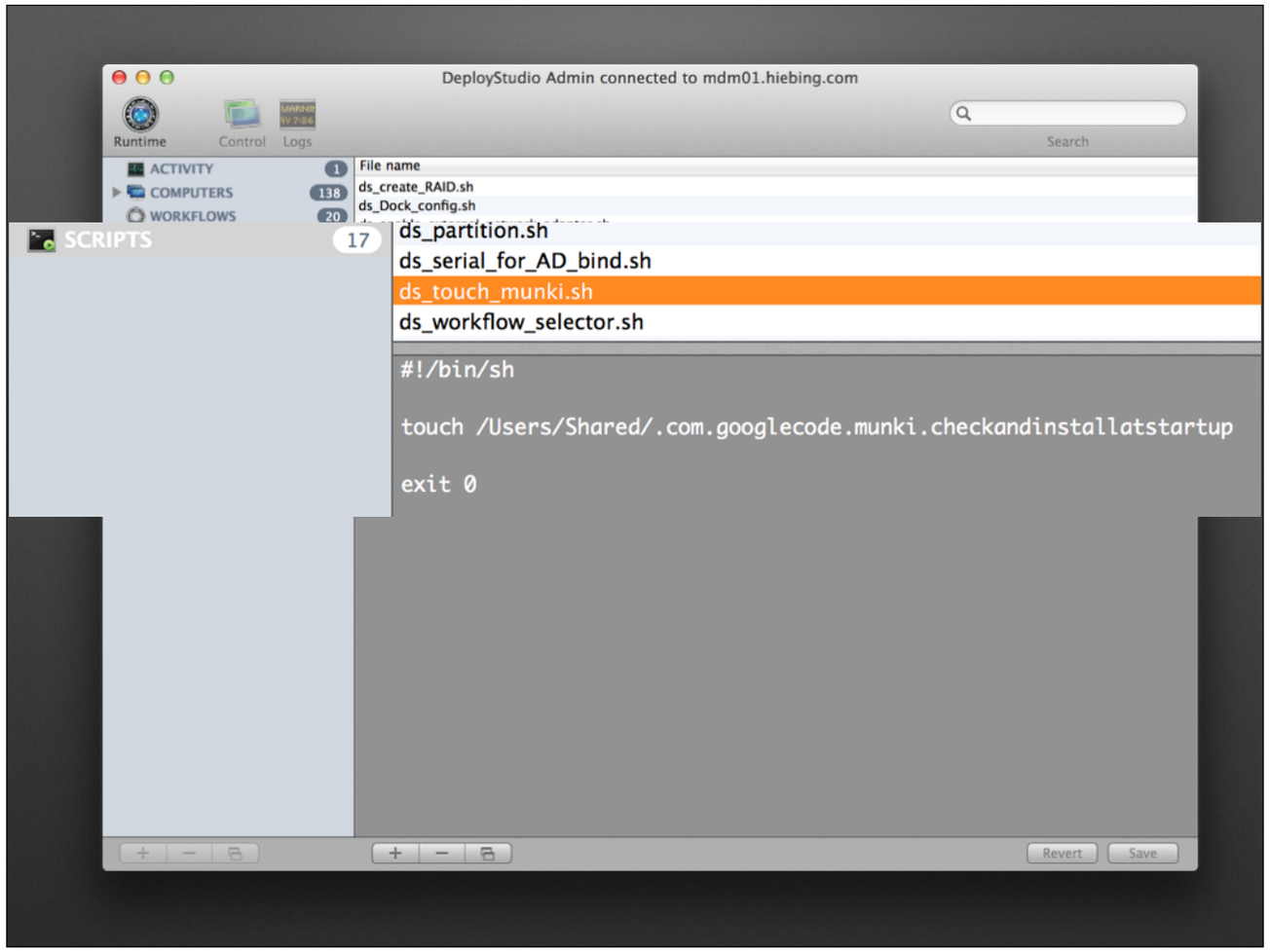
14) Preconfigure the dock

15) Run software update



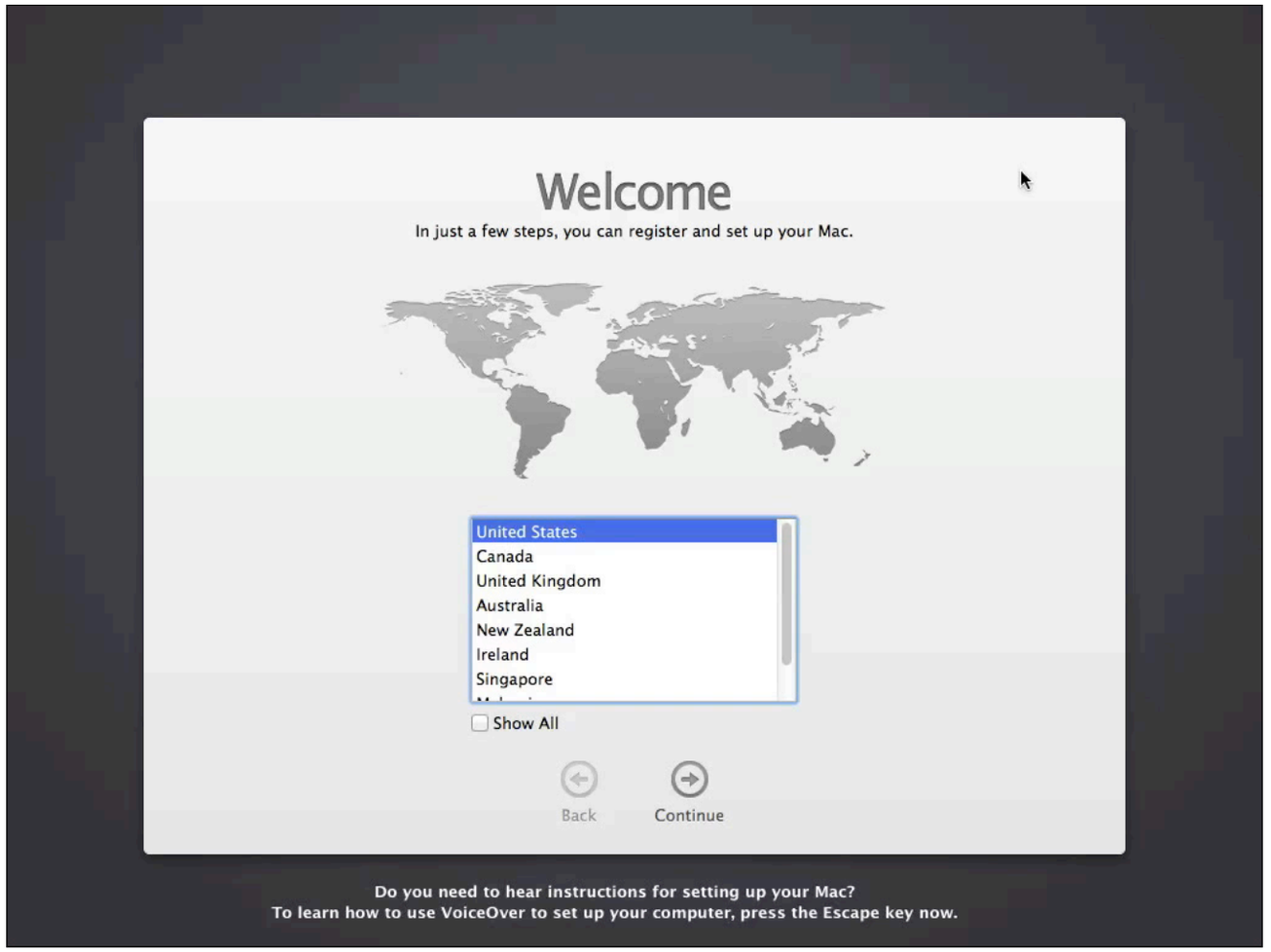
A munki launchdameon continually asks "does /Users/Shared/.com.googlecode.munki.checkandinstallatstartup exist?"

If yes, it goes into bootstrap mode, where promptly query the munki server, and install everything asked to, and keep checking. Once it comes back empty, delete the .file



A munki launchdameon continually asks "does /Users/Shared/.com.googlecode.munki.checkandinstallatstartup exist?"

If yes, it goes into bootstrap mode, where promptly query the munki server, and install everything asked to, and keep checking. Once it comes back empty, delete the .file

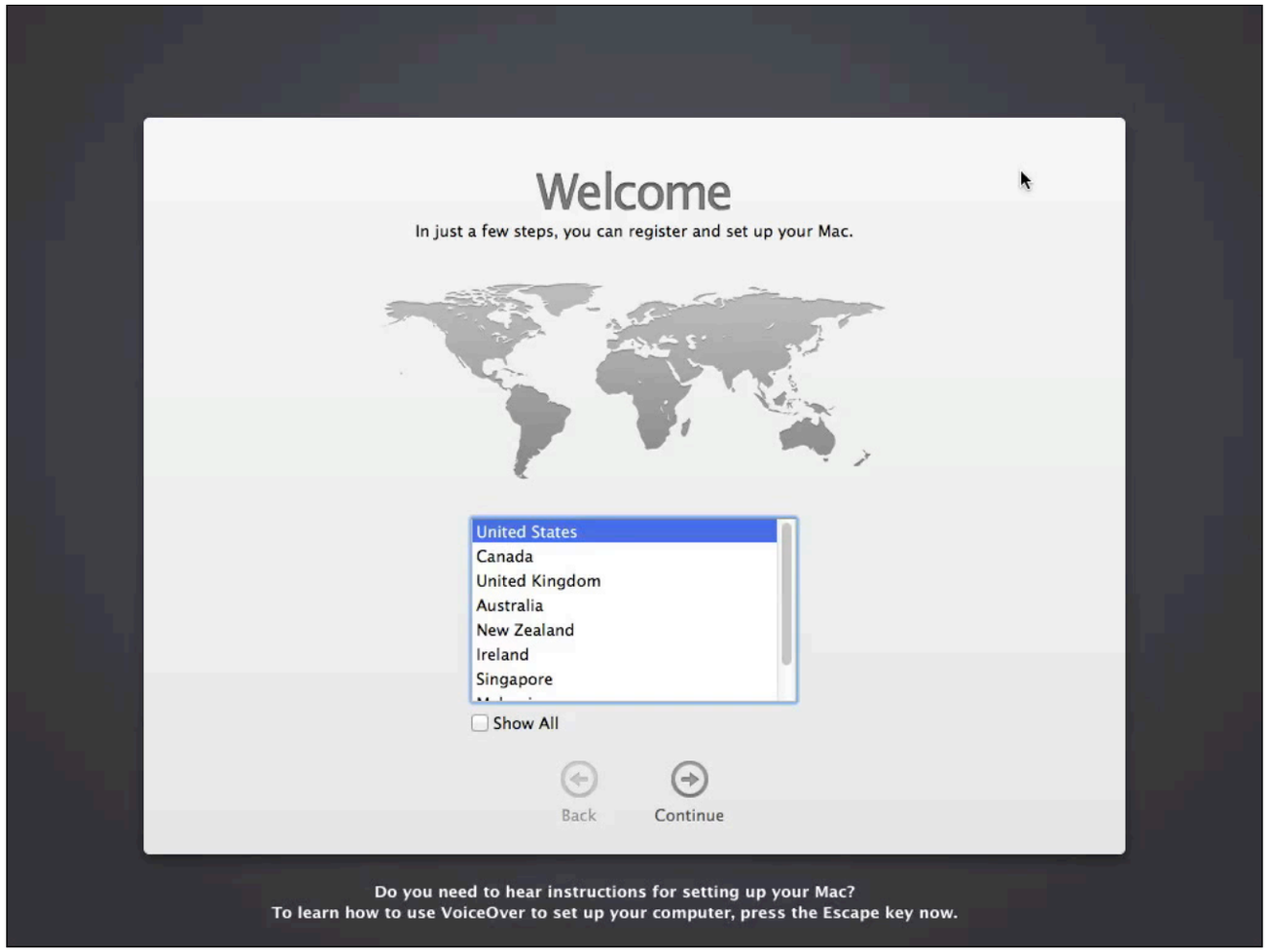


When it's all done, running a brand new machine from out of the box to configured, looks like this.

Notes:

1) This is a VM, I don't have the tools to screen record the video output. My netbooting was done by defining the network adapter as the startup disk in the VM settings- you'd boot a real machine holding down Option, or N if DeployStudio is your netboot disk.

2) Time is compressed here. thankfully



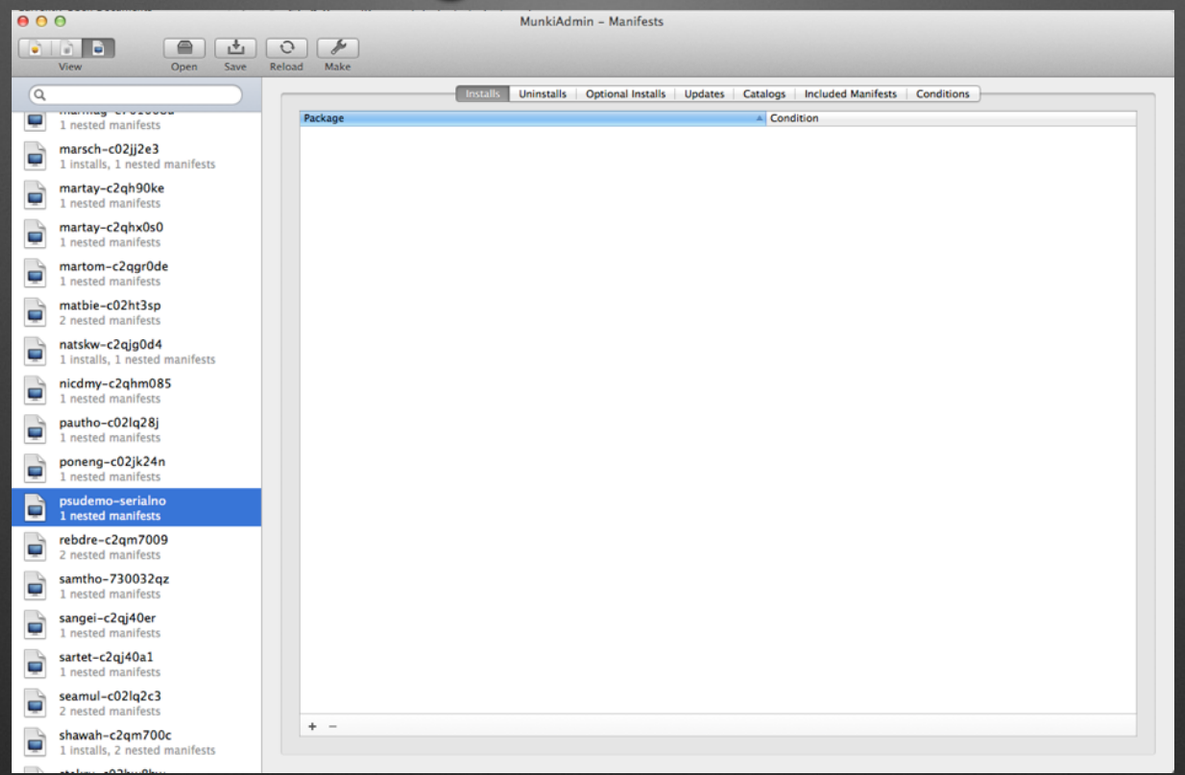
When it's all done, running a brand new machine from out of the box to configured, looks like this.

Notes:

1) This is a VM, I don't have the tools to screen record the video output. My netbooting was done by defining the network adapter as the startup disk in the VM settings- you'd boot a real machine holding down Option, or N if DeployStudio is your netboot disk.

2) Time is compressed here. thankfully

matching manifest made



Discussion!

Feedback URL:
http://j.mp/psumac47

- **Munki:** https://code.google.com/p/munki/
- **AutoPkg:** https://github.com/autopkg/autopkg
- **DeployStudio:** http://www.deploystudio.com/
- **OS X Server:** https://www.apple.com/osx/server/
- **Munkienroll:** https://github.com/edingc/munki-enroll
- **Payload-free packages:** http://managingosx.wordpress.com/2010/02/18/payload-free-package-template/
- **MunkiAdmin:** https://github.com/hjuutilainen/munkiadmin

- **Git:** http://git-scm.com/download/mac
- **AutoPkg Change Notifications script:** http://seankaiser.com/blog/2013/12/16/autopkg-change-notifications/
- **CreateUserPkg:** http://magervalp.github.io/CreateUserPkg/
- **Firstboot settings starter script:** https://github.com/rtrouton/rtrouton_scripts/tree/master/rtrouton_scripts/first_boot
- **make-profile-pkg:** https://github.com/timsutton/make-profile-pkg
- **AutoDMG:** https://github.com/MagerValp/AutoDMG
- **Payload free packages 101:** http://derflounder.wordpress.com/2014/06/01/understanding-payload-free-packages/