# Starting from Scratch with Munki

# Hi, I'm Rick!

@refreshingapathy

@rickheil

Blog/Slides: rickheil.com/psu2019

## *What We'll Cover This Morning*

- munki servers and repos
- anatomy of a munki client
- packages and pkginfos
- catalogs vs manifests
- how does munki know what to install?
- practical examples
- add-on tools
- Q&A

But first….

## *What Munki ISN'T*

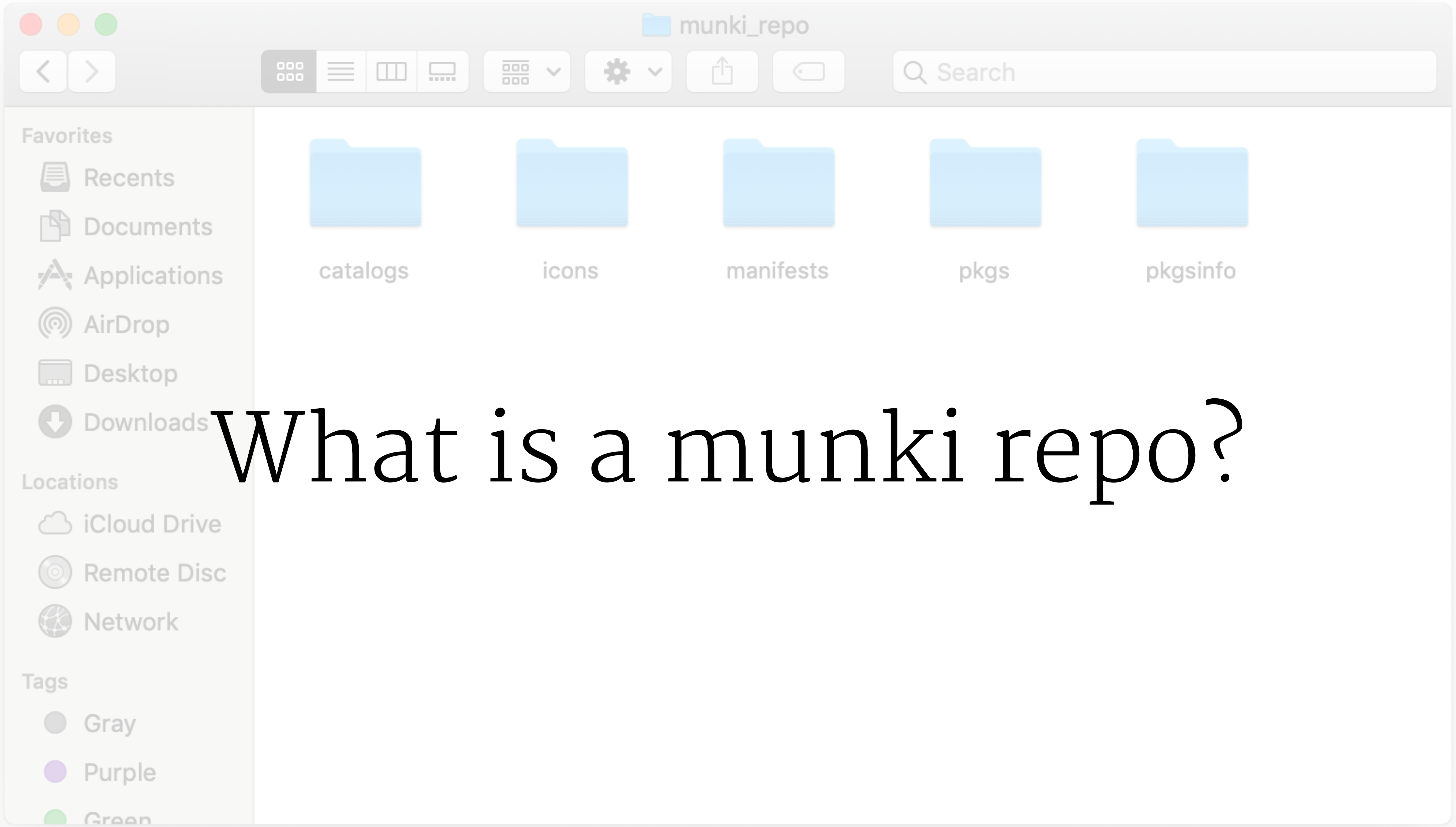
- an MDM
- a remote control /remote access tool
- reporting software
- that hard, once you get to know it

## *What Munki IS*

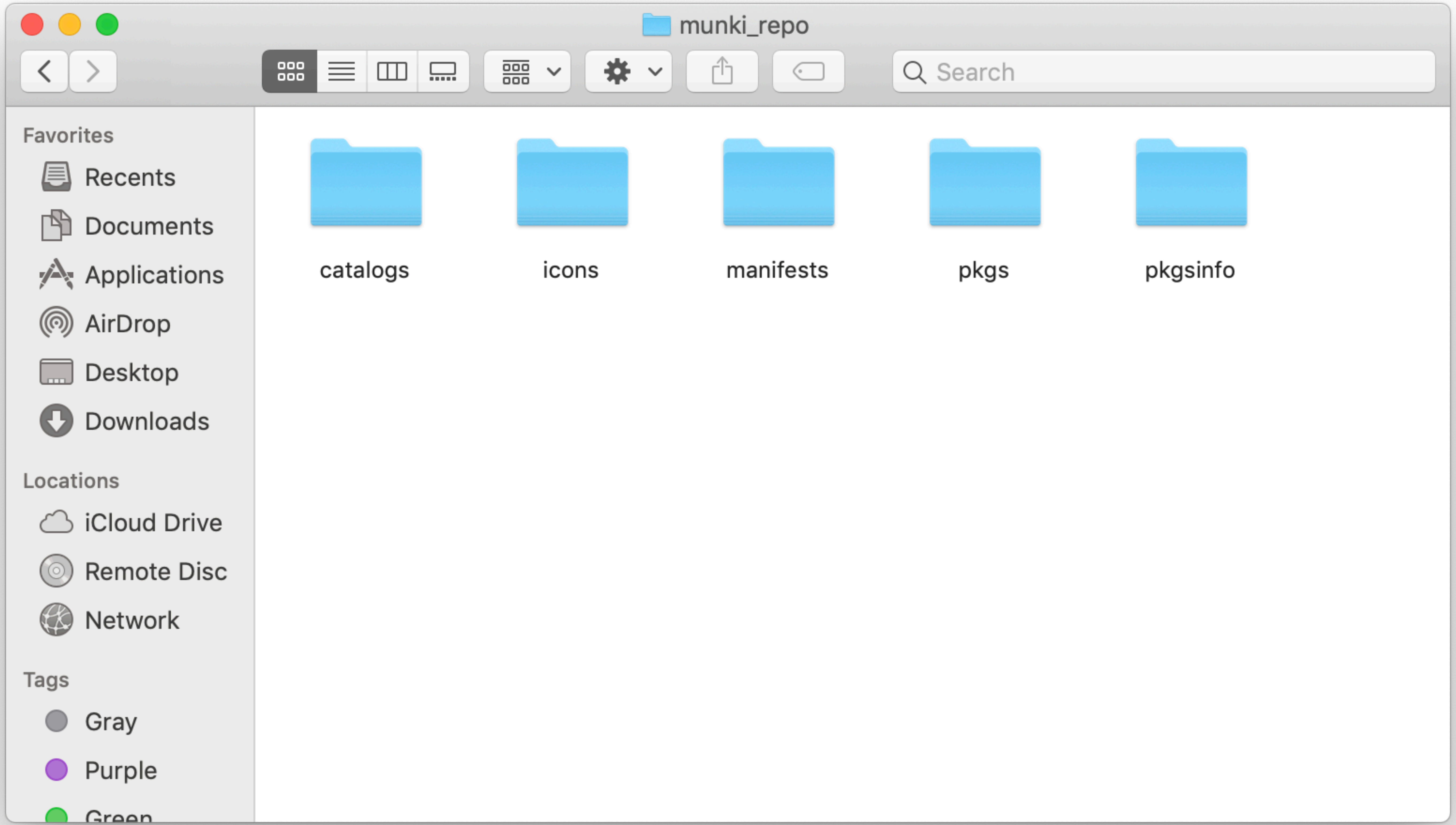
- a software management tool suite for macOS
- versatile and powerful
- well documented
- community-driven

## What is a munki server?

“The munki server is just a web server!”
~Greg



# What is a munki repo?


munki_repo
Search
Favorites
Recents
Documents
Applications
AirDrop
Desktop
Downloads
Locations
iCloud Drive
Remote Disc
Network
Tags
Gray
Purple
Green
catalogs
icons
manifests
pkgs
pkgsinfo

## A quick word on security

## What is a munki client?

- Managed Software Center (MSC)
- managedsoftwareupdate
- launchd jobs (agents and daemons)
- (plus various libraries and helper tools in /usr/local/munki)

## managedsoftwareupdate

- located in /usr/local/munki
- the "mission control" behind almost all munki functionality
- runs as root via launchd, and requires sudo when run in a terminal

## Calling managedsoftwareupdate in terminal

1. sudo /usr/local/munki/managedsoftwareupdate --auto
2. sudo /usr/local/munki/managedsoftwareupdate --checkonly
3. ⚠️ sudo /usr/local/munki/managedsoftwareupdate --installonly ⚠️

VERBOSE

```
sudo /usr/local/munki/managedsoftwareupdate -v
```

## VERBOSE

```
sudo /usr/local/munki/managedsoftwareupdate -vv
```

## VERBOSE

```
sudo /usr/local/munki/managedsoftwareupdate -vvv
```

## VERBOSE

```
sudo /usr/local/munki/managedsoftwareupdate -vvvv
```

## VERBOSE

```
sudo /usr/local/munki/managedsoftwareupdate -vvvvv
```

# Managed Software Center

## launchd jobs

- Launch Agents
  - glue between userland and the daemons
- Launch Daemons
  - scheduler
  - restart, logout, and install helpers

# managedsoftwareupdate check (LD)

```
<dict>
    <key>Label</key>
    <string>com.googlecode.munki.managedsoftwareupdate-check</string>
    <key>ProgramArguments</key>
    <array>
        <string>/usr/local/munki/supervisor</string>
        <string>--delayrandom</string>
        <string>3600</string>
        <string>--timeout</string>
        <string>43200</string>
        <string>--</string>
        <string>/usr/local/munki/managedsoftwareupdate</string>
        <string>--auto</string>
    </array>
    <key>StartCalendarInterval</key>
    <dict>
        <key>Minute</key>
        <integer>10</integer>
    </dict>
</dict>
```

## Installing the munki apps

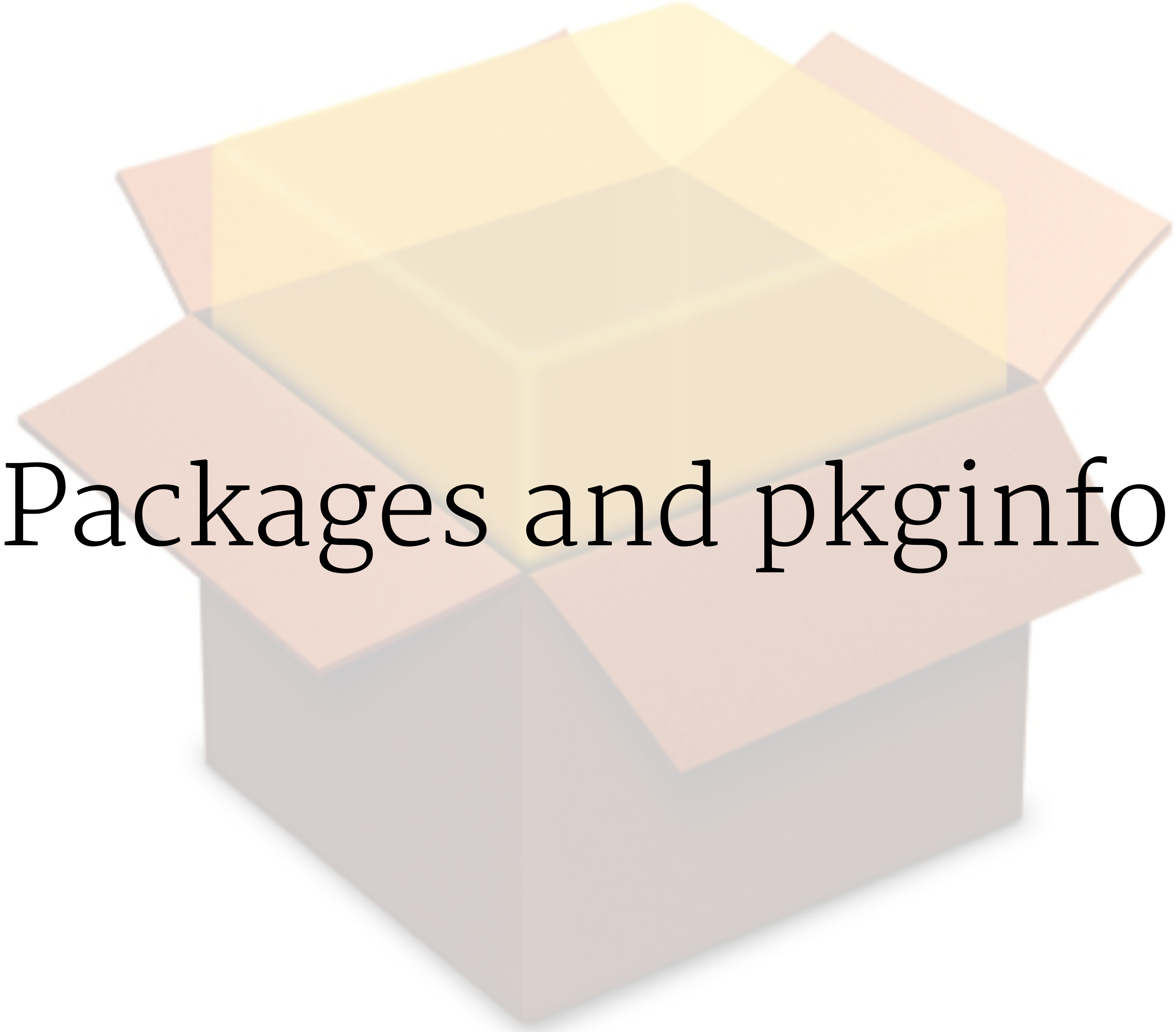
1. Download from https://github.com/munki/munki
2. Install to your machines and reboot
3. There's no step 3

## Configuring Clients

Use a configuration profile or `sudo defaults write /Library/Preferences/ManagedInstalls`

Important keys to set:
- SoftwareRepoURL
- ClientIdentifier (more on this later)

Other cool keys:
- AdditionalHttpHeaders (basic auth)
- InstallAppleSoftwareUpdates (boolean)
- LoggingLevel (integer, default of 1, higher = more logs)
- DaysBetweenNotifications



## Packages and pkginfos

## Important pkginfo keys

- version
- name vs display_name
- catalogs
- unattended install and uninstall

## More important pkginfo keys

- description
- preinstall and postinstall script
- update_for and requires
- force*install*after_date

## And two very special keys: installs and receipts

- receipts works off the pkgutil receipts database
  - check this by running `pkgutil --pkg-info PKG-ID-HERE`
  - you can get the pkg id with `pkgutil --packages` and filtering the output
  - all pkgs leave receipts, but not all receipts are useful

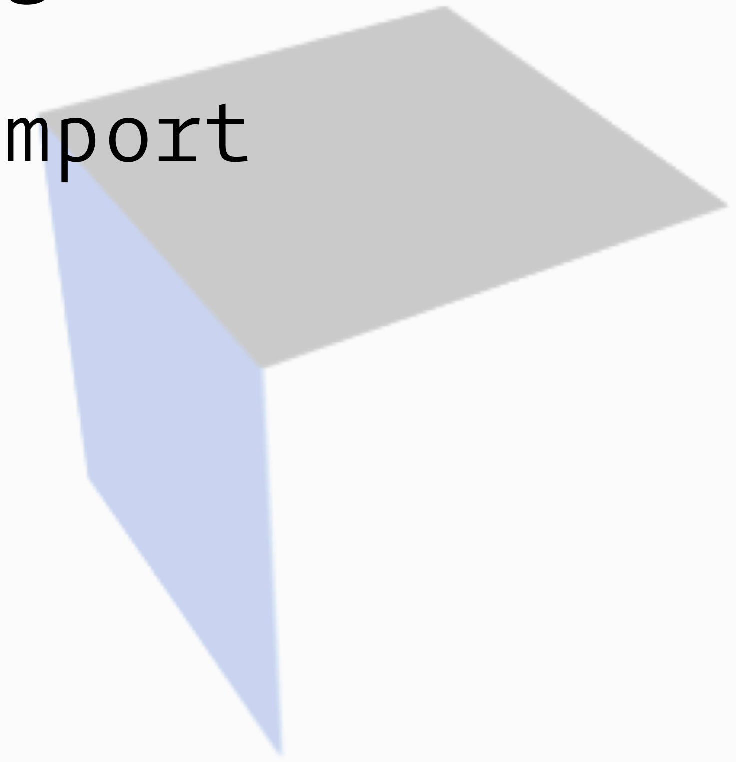
## And two very special keys: installs and receipts

- installs tell munki to look at things that are file-based to know if something is installed at a specific version
  - version strings (CFBundleShortVersionString) from Info.plist
  - file checksum / hash (MD5)
  - file presence

## Two options to make a pkginfo:

```
/usr/local/munki/makepkginfo

/usr/local/munki/munkiimport
```



## munkiimport usage: easy as pie

The first time you set up munkiimport, you will need to run the following command:

```
munkiimport --configure
```

After that, simply run `munkiimport /path/to/item.pkg` (or .dmg, .mobileconfig, etc)

I am about to show you how it's done.

## Types of things munki can install

1. standard macOS pkg
2. copy from DMG
3. configuration profile
4. nopkg
5. on demand
6. several Adobe-specific types 👎
7. macOS update metadata
8. startosinstall

## macOS pkgs + copy from DMG

- just a normal package or drag-and-drop style DMG
- munkiimport highly recommended to import
- watch for pre/post scripts in pkgs that expect to be run as users

## configuration profile

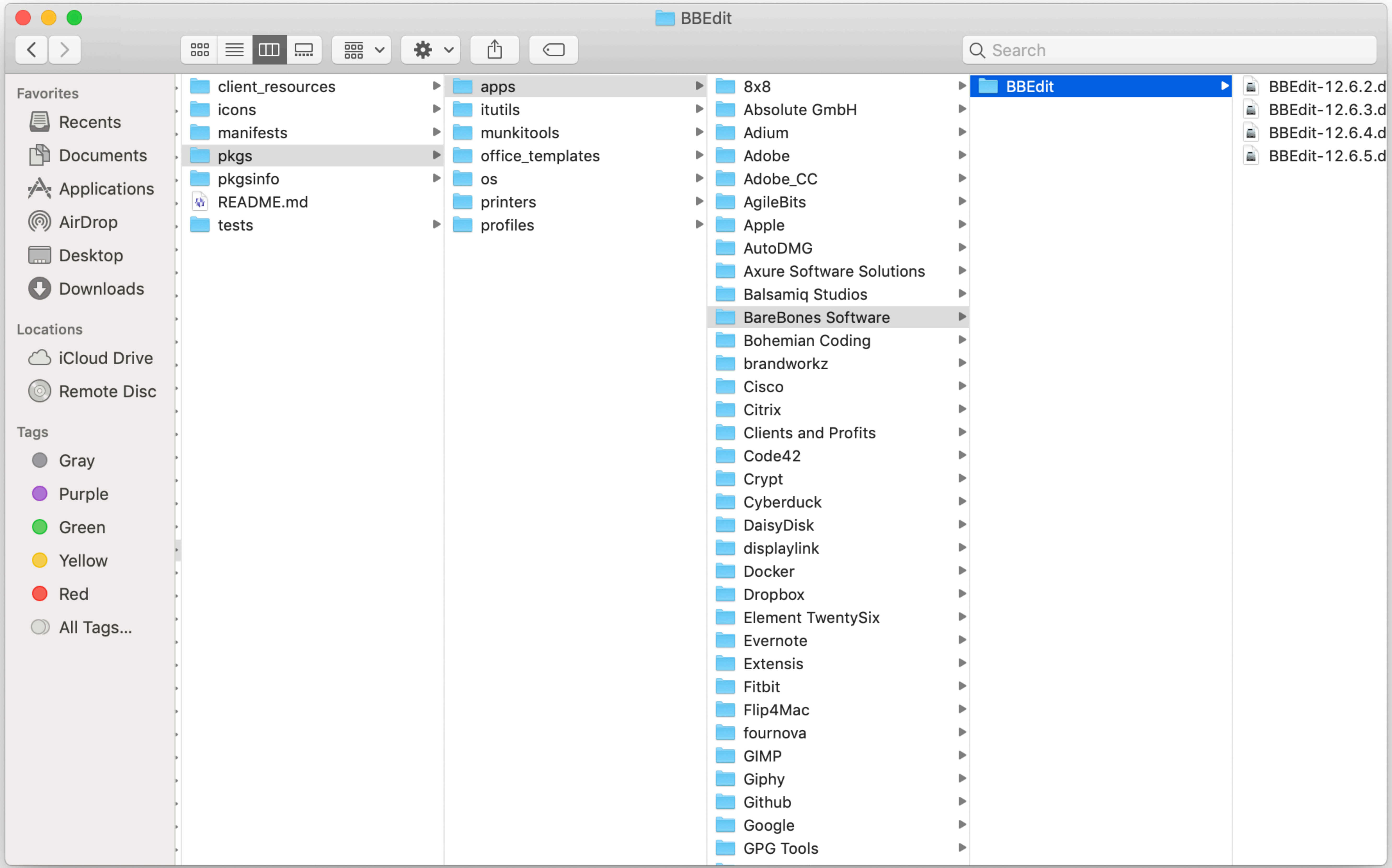
- any non-UAMDM/DEP gated preferences
- create by hand, with Profile Creator (#profilecreator on Slack), or Profile Manager (macOS Server.app)
- munkiimport highly recommended to import
- use the pkginfo "version" key to keep new versions straight (don't try to update the version key in the profile!)

## nopkg 💖

- nopkg makes my life better and can make yours better too
- uses an installcheck script to see if it needs to be installed
- uses the postinstall_script function of munki to run whatever script you need
- makepkginfo is best for this – use the `--nopkg` flag with `--installcheck_script` and `--postinstall_script` values

## A quick word about repo organization

1. Pick an organization scheme
2. Write it down
3. USE IT



client_resources
icons
manifests
pkgs
pkgsinfo
README.md
tests

apps
itutils
munkitools
office_templates
os
printers
profiles

8x8
Absolute GmbH
Adium
Adobe
Adobe_CC
AgileBits
Apple
AutoDMG
Axure Software Solutions
Balsamiq Studios
BareBones Software
Bohemian Coding
brandworkz
Cisco
Citrix
Clients and Profits
Code42
Crypt
Cyberduck
DaisyDisk
displaylink
Docker
Dropbox
Element TwentySix
Evernote
Extensis
Fitbit
Flip4Mac
fournova
GIMP
Giphy
Github
Google
GPG Tools

BBEdit

BBEdit-12.6.2.d
BBEdit-12.6.3.d
BBEdit-12.6.4.d
BBEdit-12.6.5.d

# Manifests vs Catalogs

## Catalogs

- runs through all your pkginfo files and combines them into "master files"
- created with the `makecatalogs` command
- create as many as make sense for your environment - I recommend separate test and production
- clients only read catalogs, not the pkginfo files
- not seeing a change? Make sure you ran `makecatalogs`!

## Cool catalog tricks

- munki clients evaluate catalogs in the order they are placed in the manifest
- use a second "testing" catalog to deploy new or experimental software to a test group
- separate multiple companies using the same repo (an MSP model)

## Manifests

- plist files
- instruct clients or groups of clients what to do with which software, and which catalogs to use
- clients get manifest files based on the file name

## managed_installs

- forces installation of the item
- useful for "required" software

## managed_uninstalls

- forces uninstallation of the item
- useful for "NOPE" software

## managed_updates

- » maintains software only if it was already installed
- » useful for security updates when your users have local admin rights
- » can reduce clutter of installed items versus setting everything as managed_installs

## optional_installs

- allows the user to choose whether or not they want the software installed
- allows the user to uninstall the software if they want to
- comparable to the App Store or JAMF's Self Service experiences in some ways

## included_manifests

- allows you to "nest" manifests together
- useful if you have a specific group of software that needs to be installed together

## Who am I?

Munki tries to load http(s)://SoftwareRepoURL/manifests/MANIFEST*NAME*HERE

Where MANIFEST*NAME*HERE is…

- `ClientIdentifier` if the preference is set.
- if the pref is not set, the following in this order are tried:
  1. fully qualified hostname (ricks-mac.mycompany.com)
  2. short hostname (ricks-mac)
  3. serial number
  4. site_default

## 🚨🚨 MANIFEST OPINION ALERT 🚨🚨

- one-manifest-per-machine
- group manifests
- remember: nesting is your friend

## How does munki know what to install?

Munki checks for these attributes, in this order, to evaluate whether or not to install something.

1. OnDemand (always installs)
2. installcheck_script
3. config profiles
4. installs items ("installs array")
5. receipts

## Anatomy of a (Background) Munki Run

1. Launch Daemon trigger supervisor with randomized delay
2. Supervisor calls managedsoftwareupdate

## Anatomy of a (Background) Munki Run

1. managedsoftwareupdate then:
   - reaches out to find its manifests
   - parses each item in each manifest to see whether or not it needs to be installed
   - creates a list of any installs/uninstalls, downloads any needed resources
   - performs any unattended installs or uninstalls
   - follows notification logic for any updates that need to be "attended"

How about a practical example?

## *The Problem*

- Firefox is your main supported browser, so it should be installed by default.
- You want anyone running Google Chrome to still get updates
- The developers want the open source Chromium to test with, but you don't want anyone else using it

## *Pre-Munki Solution*

- ARD
- Asking people nicely
- Walk around to each machine
- Wailing and gnashing of teeth

## *Munki-Driven Solution*

1. Import the software to your repo
2. Add software to the proper catalog(s)
3. Add to manifest(s) as needed.

## Importing + Add to Catalogs

```
munkiimport GoogleChrome.dmg
munkiimport Firefox.dmg
munkiimport Chromium.app
```

(add to catalogs during munkiimport or by editing by hand)

Don't forget to `makecatalogs`!

# Accounting and Sales Manifest

```
<dict>
    <key>catalogs</key>
    <array>
        <string>production</string>
    </array>
    <key>managed_installs</key>
    <array>
        <string>Firefox</string>
    </array>
    <key>managed_uninstalls</key>
    <array>
        <string>Chromium</string>
    </array>
    <key>managed_updates</key>
    <array>
        <string>Google Chrome</string>
    </array>
    <key>optional_installs</key>
    <array>
        <string></string>
    </array>
</dict>
```

# Developer Manifest

```
<dict>
    <key>catalogs</key>
    <array>
        <string>production</string>
    </array>
    <key>managed_installs</key>
    <array>
        <string>Firefox</string>
        <string>Chromium</string>
    </array>
    <key>managed_uninstalls</key>
    </array>
    <key>managed_updates</key>
    <array>
        <string>Google Chrome</string>
    </array>
    <key>optional_installs</key>
    <array>
        <string></string>
    </array>
</dict>
```

## Going further: autopkg

» software that automates checking for updates

» can automatically import into munki

» helps you stay on top of updates and keep things secure

https://github.com/autopkg

## Tidying up: Repoclean

- script written by Greg
- allows you to delete old versions of software that aren't used anymore
- can be helpful to spot cruft and clean things up

https://github.com/munki/munki/blob/master/code/client/repoclean

## More Advanced Tools

- MunkiReport-PHP - uses scripts on each munki run to gather info and display on a web app
- putting your repo in git / source control
- using cloud storage for your repo
- continuous integration
- application usage
- customizing MSC look and feel

## Getting Help

## *For general help and how-to questions:*

- Munki Wiki (https://github.com/munki/munki/wiki)
- MacAdmins Slack, #munki (sign up at macadmins.org)
- Join the Munki-Discuss mailing list (Google Groups)

## *For code questions or reporting bugs:*

- MacAdmins Slack, #munki (sign up at macadmins.org)
- Post a Github issue (github.com/munki/munki)

## Q&A

THANK YOU!