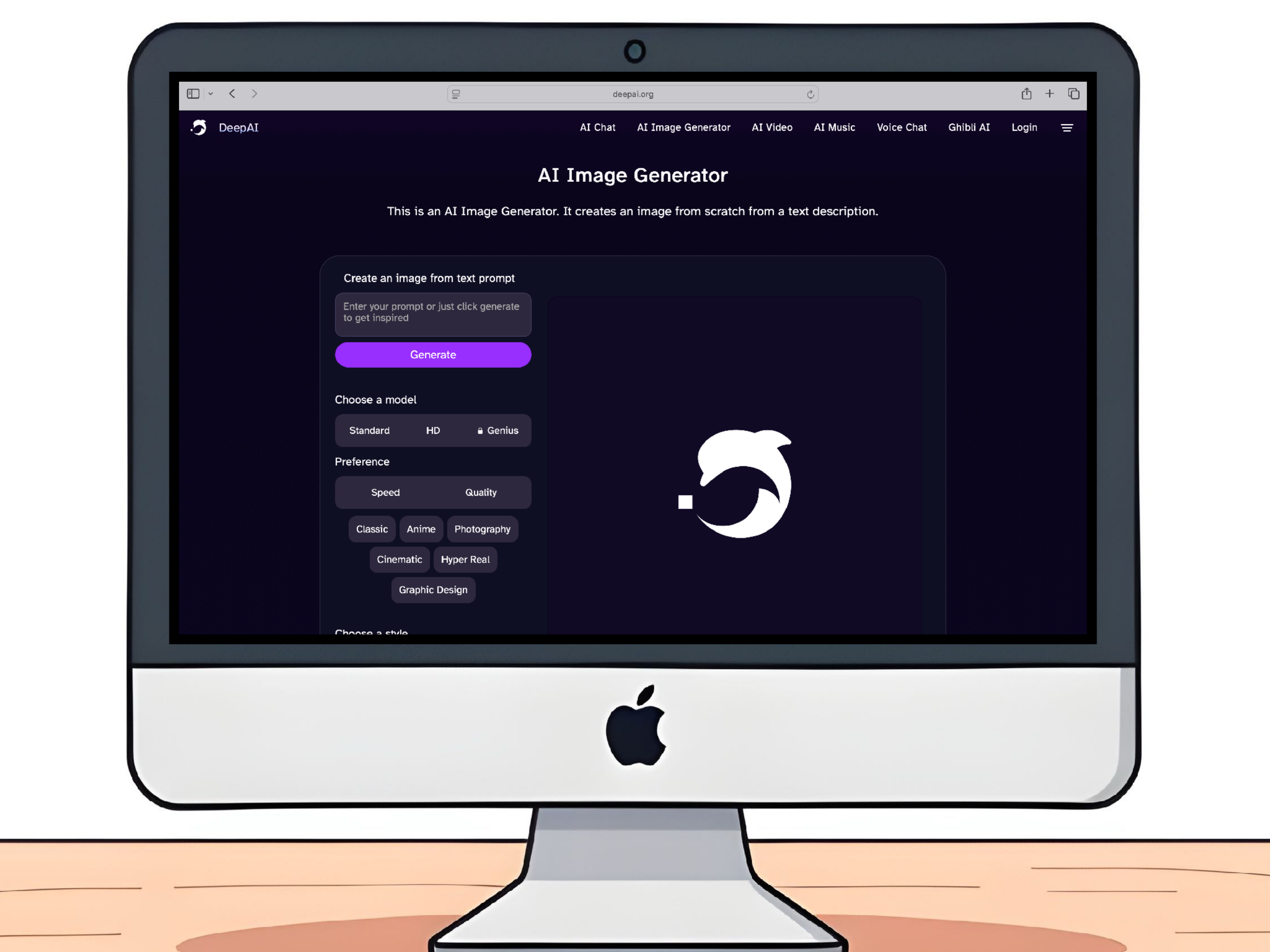
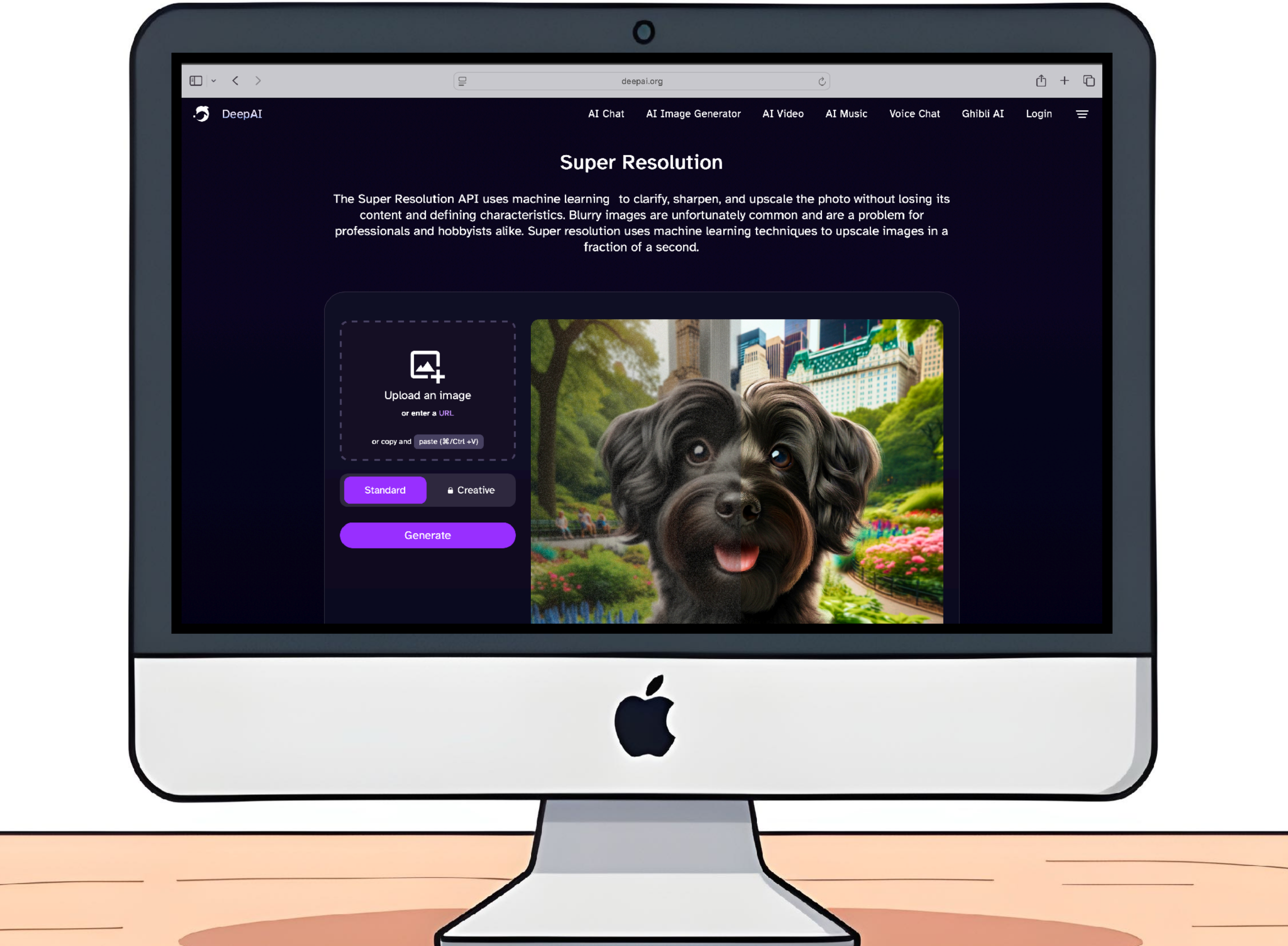
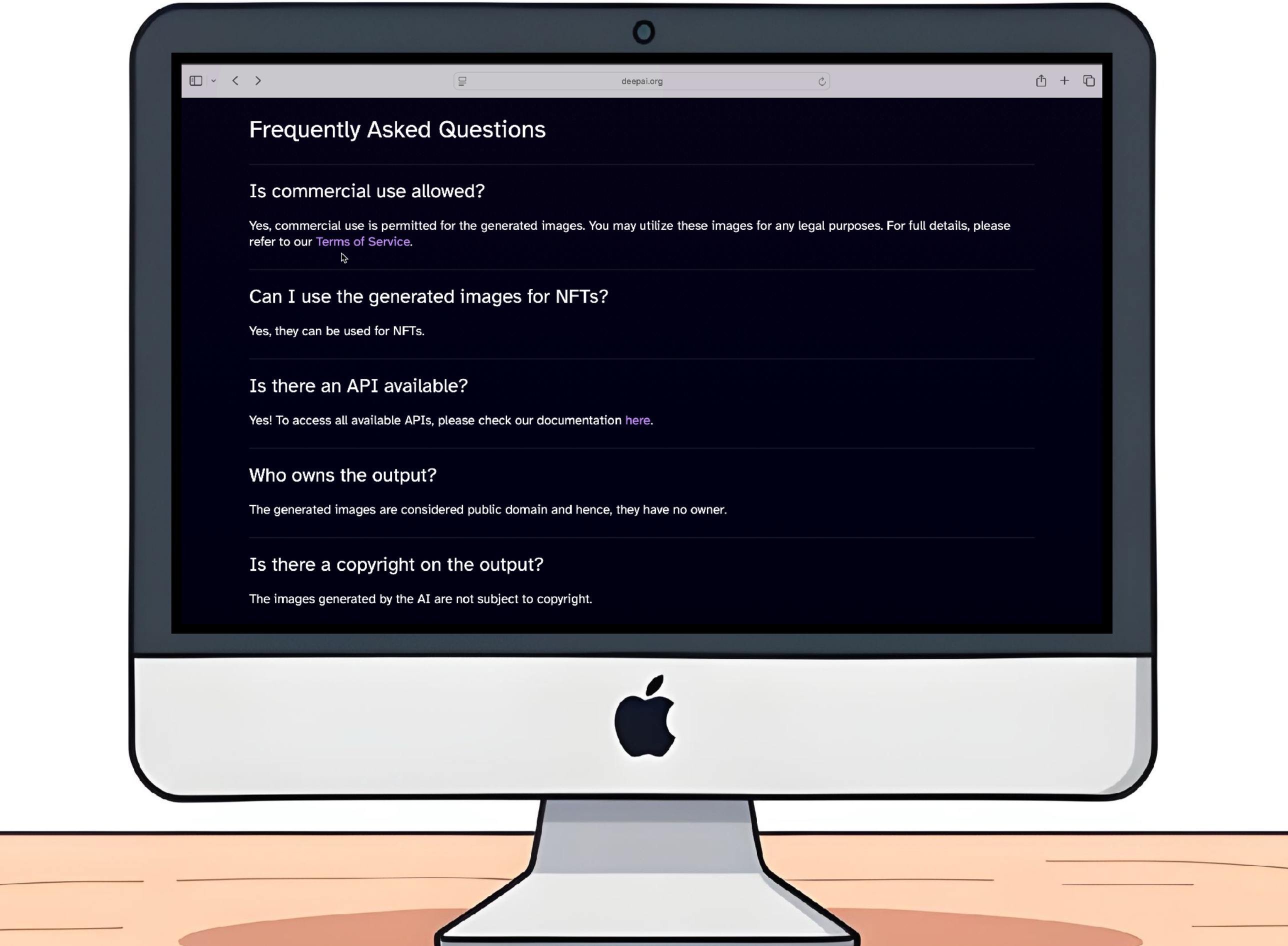
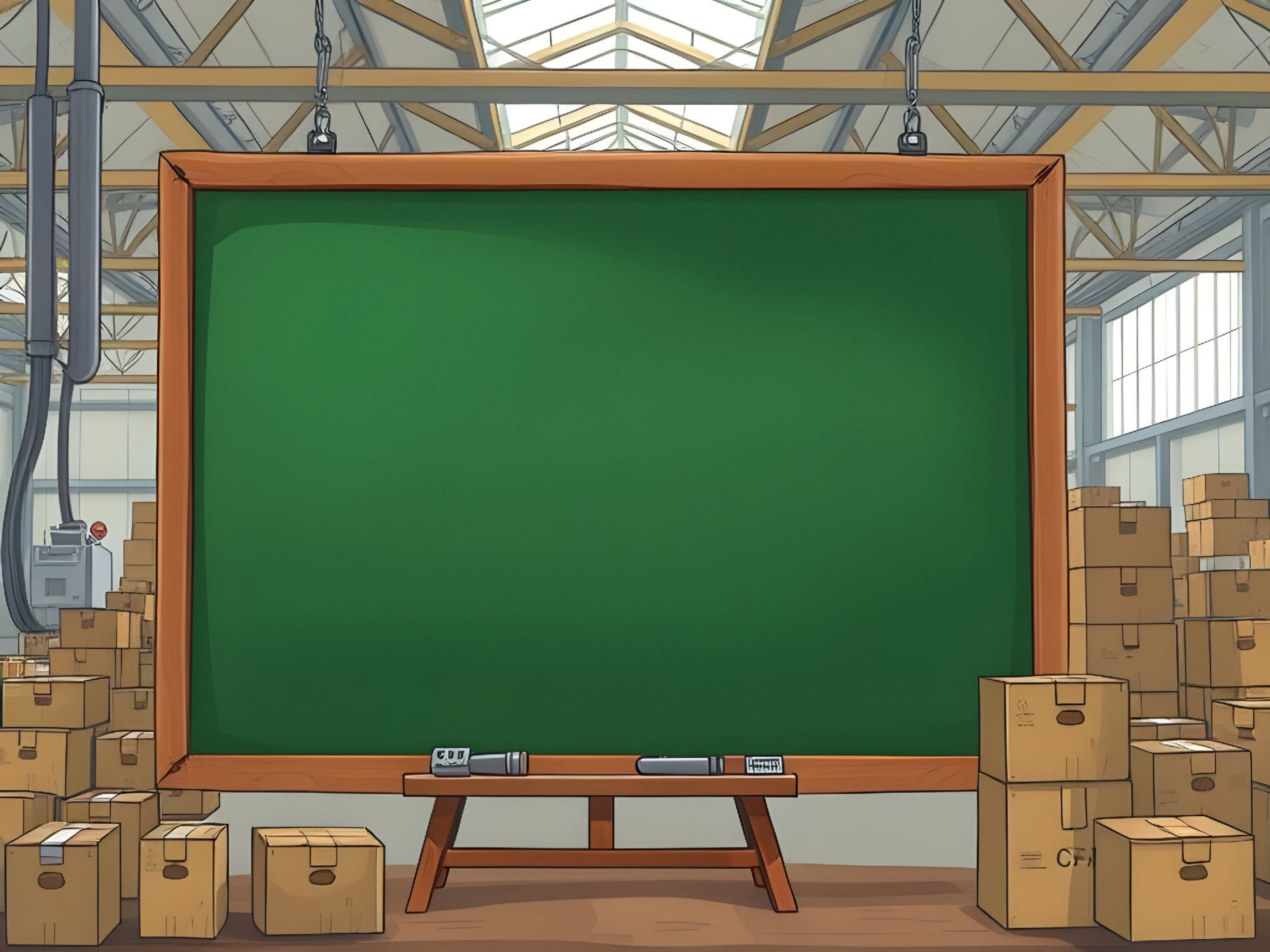
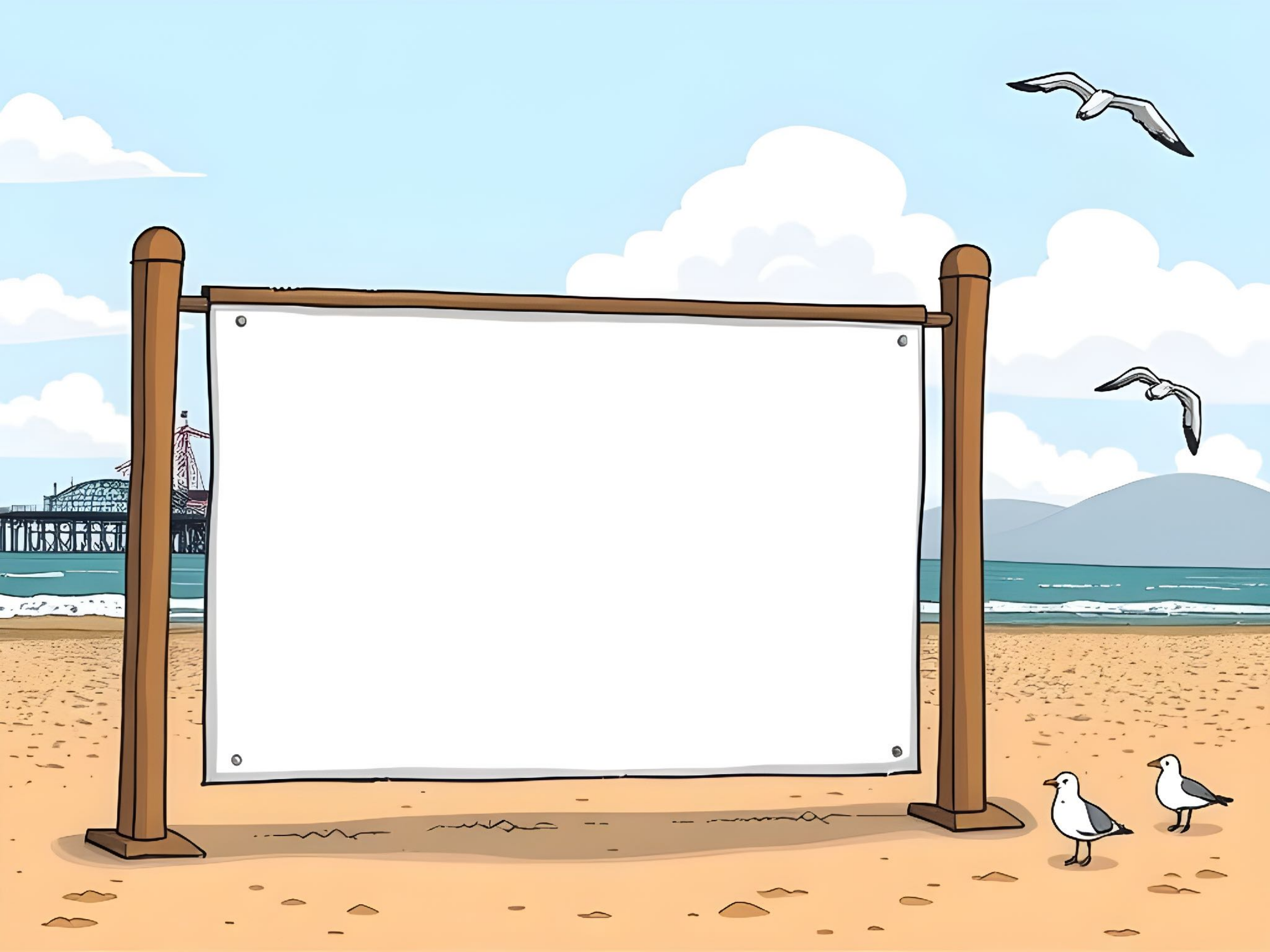
is for packages.


deepai.org
DeepAI
AI Chat
AI Image Generator
AI Video
AI Music
Voice Chat
Ghibli AI
Login
AI Image Generator
This is an AI Image Generator. It creates an image from scratch from a text description.
Create an image from text prompt
Enter your prompt or just click generate to get inspired
Generate
Choose a model
Standard
HD
Genius
Preference
Speed
Quality
Classic
Anime
Photography
Cinematic
Hyper Real
Graphic Design


deepai.org
DeepAI
AI Chat
AI Image Generator
AI Video
AI Music
Voice Chat
Ghibli AI
Login
Super Resolution
The Super Resolution API uses machine learning to clarify, sharpen, and upscale the photo without losing its content and defining characteristics. Blurry images are unfortunately common and are a problem for professionals and hobbyists alike. Super resolution uses machine learning techniques to upscale images in a fraction of a second.
Upload an image
or enter a URL
or copy and paste (⌘/Ctrl +V)
Standard
Creative
Generate

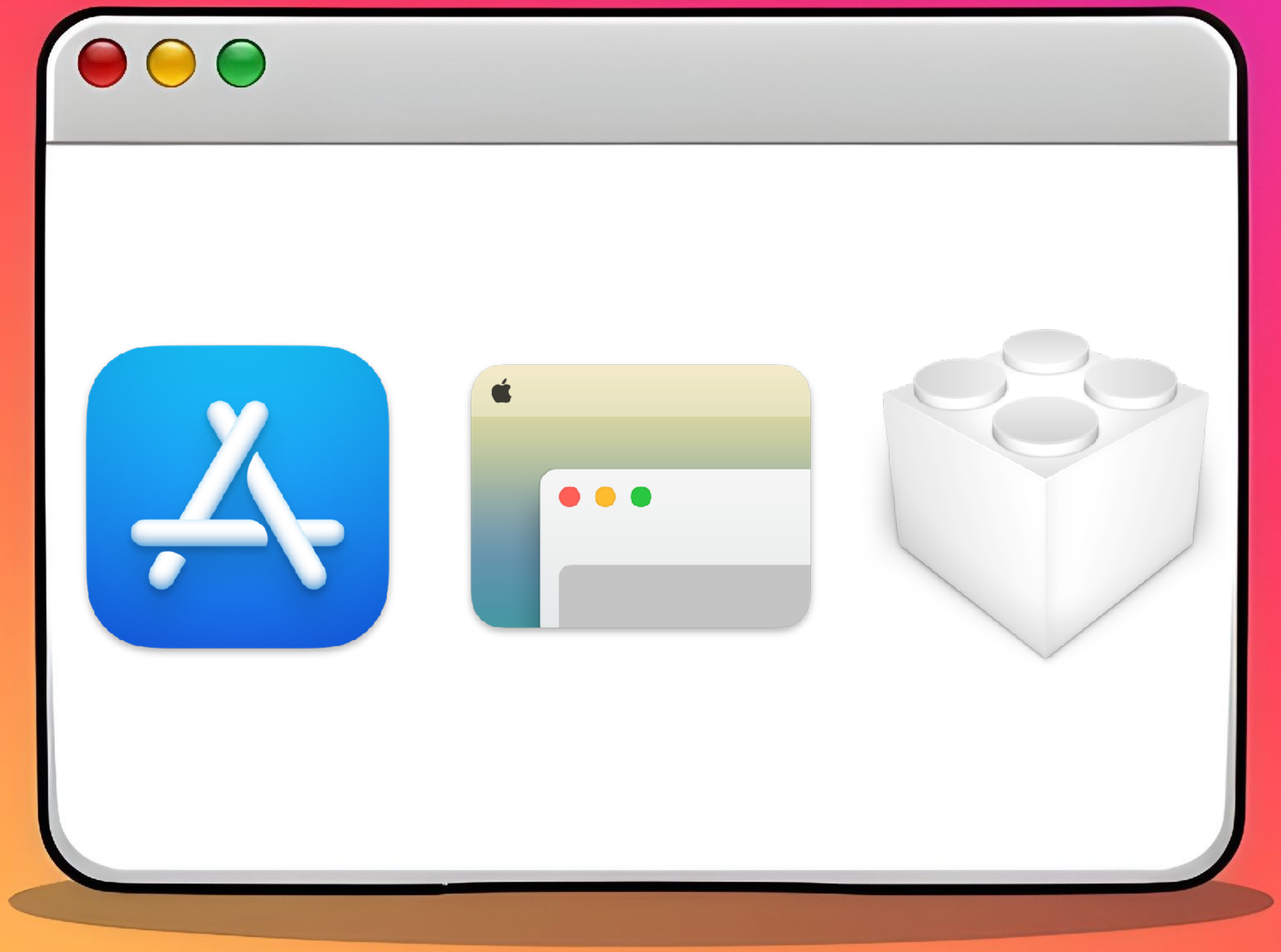

deepai.org
Frequently Asked Questions
Is commercial use allowed?
Yes, commercial use is permitted for the generated images. You may utilize these images for any legal purposes. For full details, please refer to our Terms of Service.
Can I use the generated images for NFTs?
Yes, they can be used for NFTs.
Is there an API available?
Yes! To access all available APIs, please check our documentation here.
Who owns the output?
The generated images are considered public domain and hence, they have no owner.
Is there a copyright on the output?
The images generated by the AI are not subject to copyright.

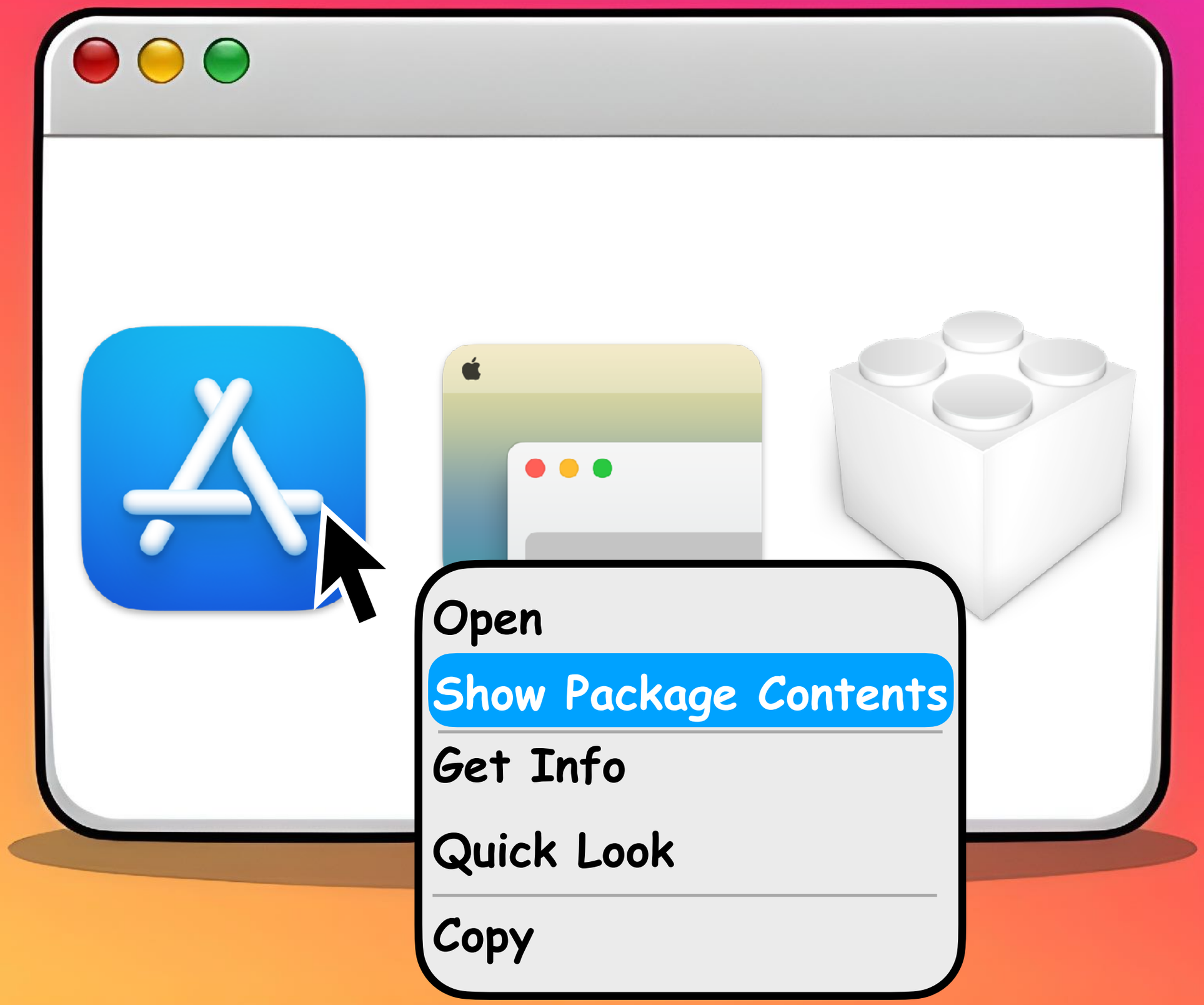


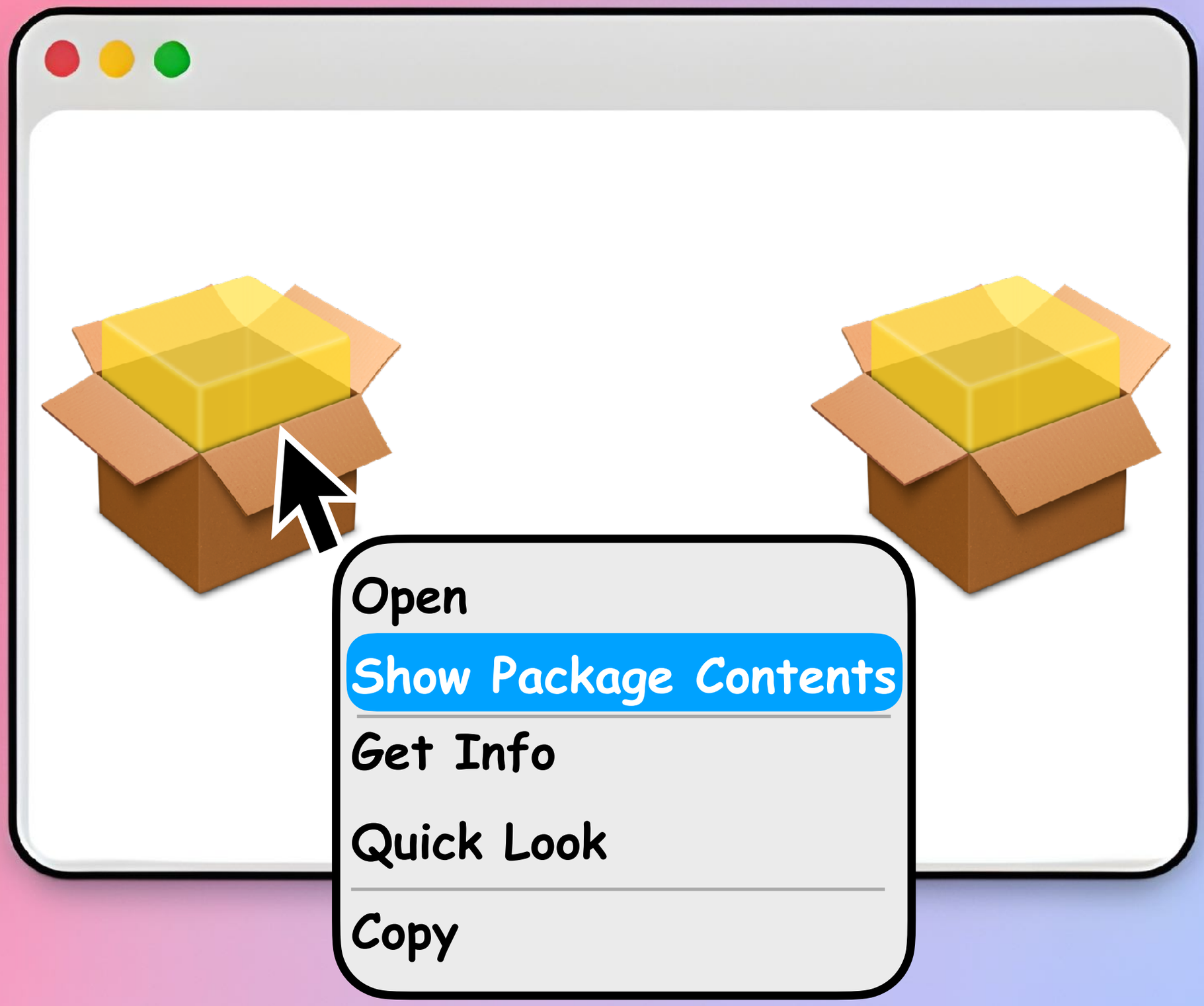


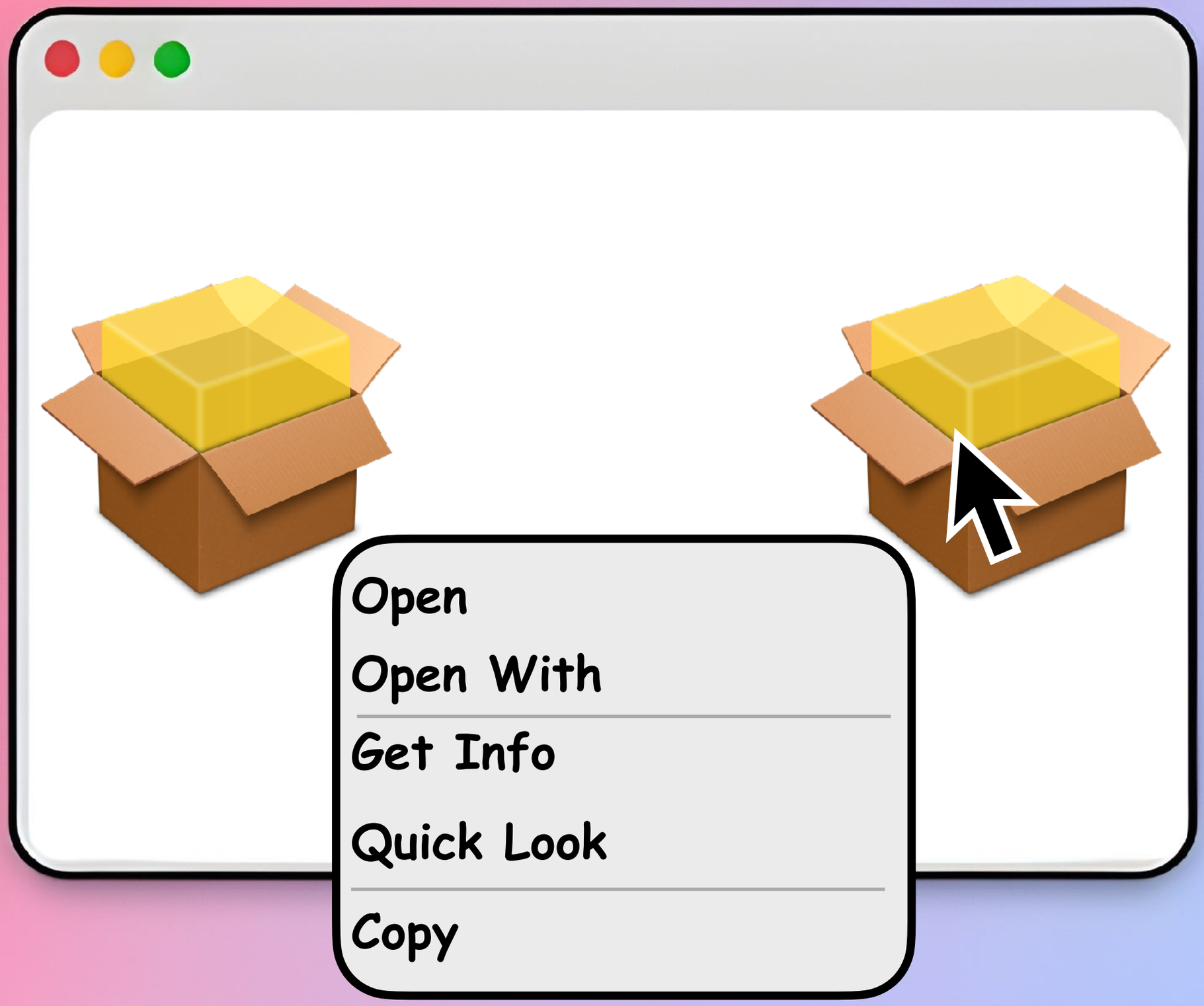
What is a package?
Inspecting packages
Creating packages
Installing packages
Uninstalling packages

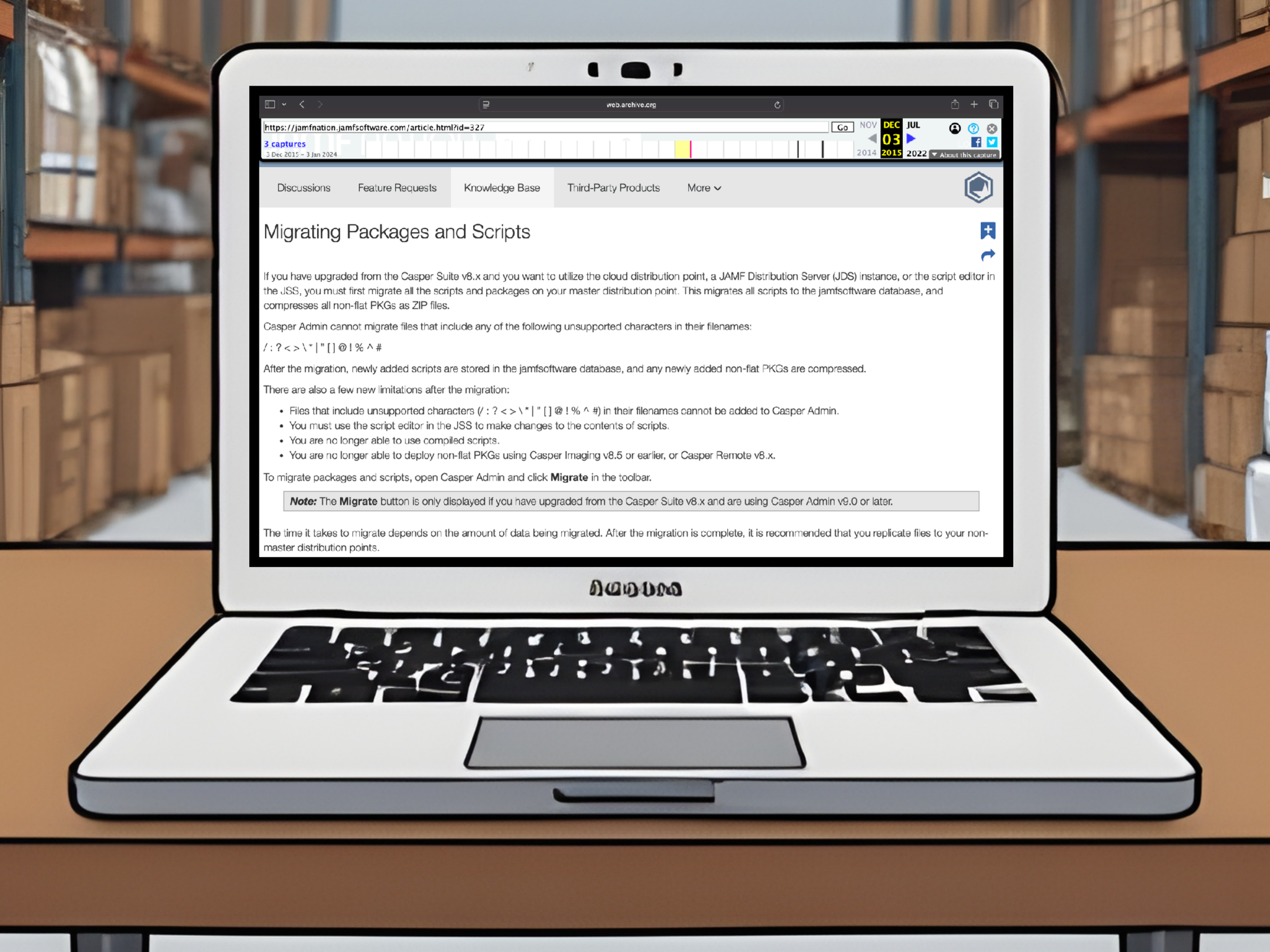
What is a package?




Open
Show Package Contents
Get Info
Quick Look
Copy


Open
Show Package Contents
Get Info
Quick Look
Copy


Open
Open With
Get Info
Quick Look
Copy



web.archive.org

https://jamfnation.jamfsoftware.com/article.html?id=327 Go

3 captures
3 Dec 2015 - 3 Jan 2024

NOV 2014 DEC 03 2015 JUL 2022 About this capture

Discussions | Feature Requests | Knowledge Base | Third-Party Products | More

Migrating Packages and Scripts

If you have upgraded from the Casper Suite v8.x and you want to utilize the cloud distribution point, a JAMF Distribution Server (JDS) instance, or the script editor in the JSS, you must first migrate all the scripts and packages on your master distribution point. This migrates all scripts to the jamfsoftware database, and compresses all non-flat PKGs as ZIP files.

Casper Admin cannot migrate files that include any of the following unsupported characters in their filenames:

/ : ? < > \ * | " [ ] @ ! % ^ #

After the migration, newly added scripts are stored in the jamfsoftware database, and any newly added non-flat PKGs are compressed.

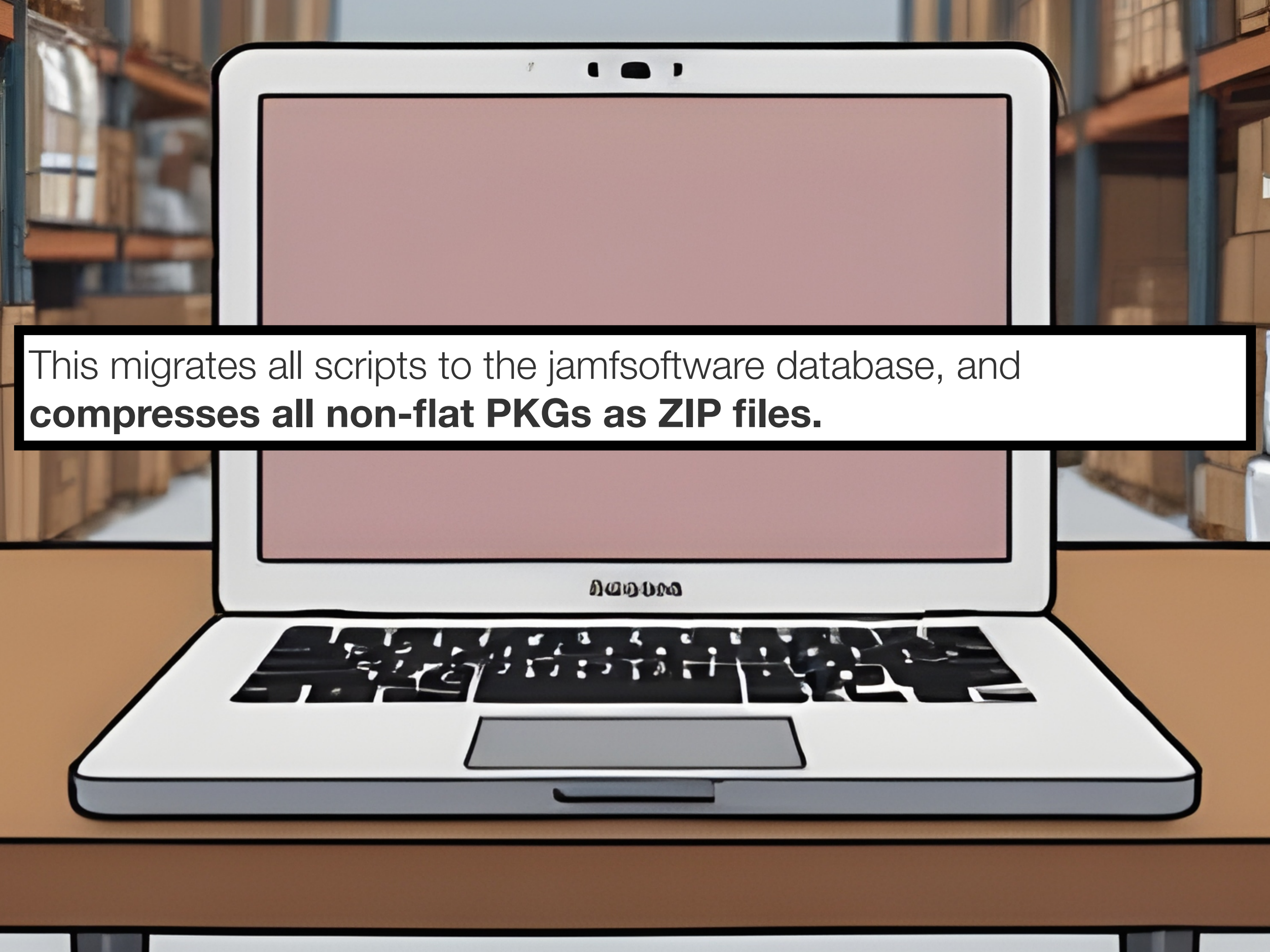
There are also a few new limitations after the migration:

- Files that include unsupported characters (/ : ? < > \ * | " [ ] @ ! % ^ #) in their filenames cannot be added to Casper Admin.
- You must use the script editor in the JSS to make changes to the contents of scripts.
- You are no longer able to use compiled scripts.
- You are no longer able to deploy non-flat PKGs using Casper Imaging v8.5 or earlier, or Casper Remote v8.x.

To migrate packages and scripts, open Casper Admin and click **Migrate** in the toolbar.

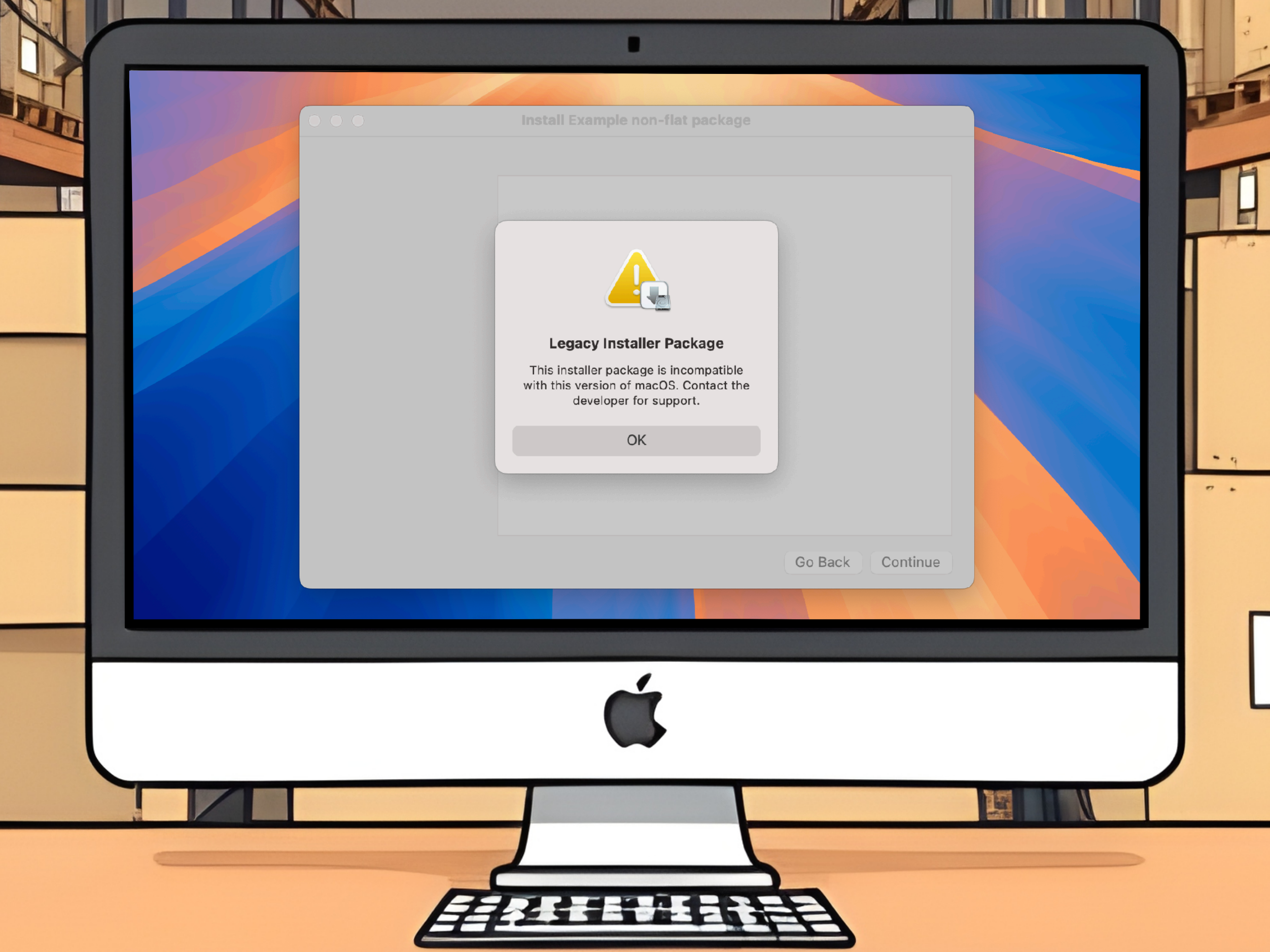
> ***Note:*** The **Migrate** button is only displayed if you have upgraded from the Casper Suite v8.x and are using Casper Admin v9.0 or later.

The time it takes to migrate depends on the amount of data being migrated. After the migration is complete, it is recommended that you replicate files to your non-master distribution points.



This migrates all scripts to the jamfsoftware database, and **compresses all non-flat PKGs as ZIP files.**

Install Example non-flat package
Legacy Installer Package
This installer package will be incompatible with a future version of macOS. Contact the developer for support.
OK
Go Back
Continue


Install Example non-flat package
Legacy Installer Package
This installer package is incompatible with this version of macOS. Contact the developer for support.
OK
Go Back
Continue

When is a package not a package?

When is a package not a package?

When its a package!

mothersruin.com

Package Format	Description
Distribution Archive	A *distribution archive* (sometimes also called a *distribution package*) combines one or more component packages with a Distribution file in a single flat file, using the xar archive format. A distribution archive can also be signed. Because it is self-contained, a distribution archive doesn't require a disk image for delivery, although one might be used anyway — perhaps to include documentation or fancy backgroud images. The distribution archive is probably the most common format these days.
Product Archive	A *product archive* is a distribution archive in which the Distribution file declares a *product identifier and version*. The `productbuild(1)` man page terms this a product archive. (Note that, if you use `productbuild` to create a product archive, the `--identifier` and `--version` options are generally required, unless a hand-written Distribution file is given with the `--distribution` option.) A product archive is required for submission to the App Store (and is also distributed **by** the App Store, after having been re-signed by Apple). A product archive is also required by the `--installpackage` option of the `startosinstall` command (bundled in the "Install macOS" app).
Distribution Archive with External Components	Although a distribution archive usually carries its component packages in the same xar archive file, it is *possible* for the Distribution file to reference component packages in other places (such as in a sibling folder). This format is uncommon, although Apple still uses it occasionally. By its very nature, this package format is **not** self-contained, and thus will usually be delivered inside a disk image or other container.
Component Package	Although a component package is typically only a building block for the distributed installer package, it might also be distributed by itself. A stand-alone component package will behave like most other packages when opened in the macOS Installer. The traditional macOS Software Update mechanism also uses stand-alone component packages. (In Software Update, the Distribution files are evaluated first, to determine which updates are applicable, so these must be downloaded separately from the component packages they reference.) A stand-alone component package is also a flat file — using the xar archive format — and is also typically signed.
macOS Full Installer	This is a special type of package that Apple uses to deliver the "Install macOS" app, which is used to upgrade to a new major version of macOS. You'll see the version of macOS that it would install further down on the Package Info tab. See more than you wanted to know about this below.
Distribution Metapackage (Bundle)	Prior to the Mac OS X 10.5 (Leopard), the *distribution metapackage* served the same purpose as today's distribution archive. It is a bundle-style package that combines combines one or more component packages with a Distribution file. The component packages might be inside or outside of the metapackage bundle. Since a bundle-style package is a directory, rather than a flat file, it typically requires a disk image or other container for delivery. Also note that bundle-style packages can **not** be signed. Historically, distribution metapackages used a distinct *.mpkg* extension, but these days, it doesn't really matter.
Component Package (Bundle)	As with flat packages, stand-alone bundle-style component packages are also possible. As with distribution metapackages, this format generally requires a disk image or other container, and likewise can **not** be signed.
Info.plist Metapackage (Bundle)	Prior to the introduction of the Distribution file in Mac OS X 10.4 (Tiger), metapackages were possible, but used an Info.plist to reference the component packages. We really hope that no one is making these any more, but some people like to party like it's 2006.

> We are not Mac admins ourselves, so are being purposely vague about the specifics of deployment. For a good introduction to the different package formats and the implications for deployment, check out Armin Briegel's talk The Encylopaedia of Packages from MacDevOpsYVR 2021.

Unfortunately, there isn't much standardization on what to *call* the different package formats — Apple presents them all as simply “installer packages” and doesn't offer any distinction. So we've tried to name them based on available clues, some community practice, and our own personal experience working on the things. So that you know what we mean by something like Distribution Archive, the table below gives more background on each of the possible formats:

Not Secure — s.sudre.free.fr

Flat Package Format - The missing documentation

Description

This document will try to document the Flat Package Format introduced by Apple in Mac OS X 10.5. Please keep in mind, this is built from reverse-engineering the format so it may contain inaccuracies for options that are both undocumented and kept secret by Apple.

The world is flat, deal with it.

With previous versions of Mac OS X, a package was a Mac OS X bundle that contains a payload, scripts and a description of the package requirements and behavior. This type of package is still supported in Mac OS X 10.5 but a new format of package was introduced: flat packages.

Flat packages are called that way supposedly because the package is now just a file. This file is actually a xar archive.

You can inspect the contents of a flat package 3 ways:

- You can open a flat package using the Flat Package Editor application. This will allow you to see which files are within the archive and inspect the contents of some of these files.

  This application is within the PackageMaker application bundle so you need to have installed the Developer Tools. Be aware this application is unfinished and buggy.

- You can list the contents of a flat package using the xar tool. You can also extract files from the package using this tool.

  The latest version of xar at the time of this writing is 1.5.2 and it does not compile correctly on Mac OS X 10.5. The latest available version of the xar source code used by Apple is available on the Apple Open Source website.

- You can use Pacifist to see the files from the package, extract the payload or a specific file from the payload.

A flat package can describe either a metapackage or a single package. We won't discuss metapackage in this document for the time being. Here are 3 examples of flat packages:

testFolderEmpty

testPermScripts

Not Secure — s.sudre.free.fr

Flat Package Format - The missing documentation

Description

This document will try to document the Flat Package Format introduced by Apple in Mac OS X 10.5. Please keep in mind, this is built from reverse-engineering the format so it may contain inaccuracies for options that are both undocumented and kept secret by Apple.

The world is flat, deal with it.

With previous versions of Mac OS X, a package was a Mac OS X bundle that contains a payload, scripts and a description of the package requirements and behavior. This type of package is still supported in Mac OS X 10.5 but a new format of package was introduced: flat packages.

Flat packages are called that way supposedly because the package is now just a file. This file is actually a xar archive.

The world is flat, deal with it.

What is a package?

Decide on a container format

To get started, decide on your container format. Mac products support two distribution channels:

- You can distribute apps in the Mac App Store.
- You can use Developer ID signing to distribute apps and other products directly, outside of the Mac App Store.

To distribute an app in the Mac App Store, you submit the app as an installer package. For direct distribution, you have a choice of various container formats, the most common being:

Zip archive (`.zip`)
: You can't sign a zip archive, so any files or folders you include that aren't covered by your code signature may be tampered with by an attacker. The person receiving your zip archive opens it with Finder to unarchive the contents, which they optionally move into their preferred location.

Disk image (`.dmg`)
: You can sign a disk image, which protects all files and folders you include from modification after you sign it. The person receiving your disk image opens it in Finder to access its contents, and they may choose to run your app from the disk image or move it to their preferred location. This experience is easiest if your product is a single file or bundle.

Installer package (`.pkg`)
: You need to sign an Installer package, which protects all files and folders you include from modification after you sign it. The person receiving your Installer package opens it in Finder to launch the Installer app, which guides them through the steps needed to install your product. An installer package is the best choice if your product contains multiple components, must be copied to specific locations, or if you need to run custom code during installation.

Decide on a container format

To get started, decide on your container format. Mac products support two distribution channels:

- You can distribute apps in the Mac App Store.
- You can use Developer ID signing to distribute apps and other products directly, outside of the Mac App Store.

To distribute an app in the Mac App Store, you submit the app as an installer package. For direct

Installer package (`.pkg`)

You need to sign an Installer package, which protects all files and folders you include from modification after you sign it. The person receiving your Installer package opens it in Finder to launch the Installer app, which guides them through the steps needed to install your product. An installer package is the best choice if your product contains multiple components, must be copied to specific locations, or if you need to run custom code during installation.

Installer package (`.pkg`)

You need to sign an Installer package, which protects all files and folders you include from modification after you sign it. The person receiving your Installer package opens it in Finder to launch the Installer app, which guides them through the steps needed to install your product. An installer package is the best choice if your product contains multiple components, must be copied to specific locations, or if you need to run custom code during installation.

Notarize your product

If you distribute your product directly, notarize the file you intend to distribute to your users. For more information, see Customizing the notarization workflow. Skip the “Export a package for notarization” section because you already have the file that you want to submit.

If you distribute your product using nested containers, only notarize the outermost container. For example, if you have an app inside an installer package on a disk image, sign the app, sign the installer package, and sign the disk image, but only notarize the disk image.

If you use a third-party installer tool, refer to Customizing the notarization workflow.

Staple your product

Once you’ve notarized your product, staple the resulting ticket to the file you intend to distribute. For information about how to do this for an app within a zip archive, see “Staple the Ticket to Your Distribution” in Customizing the notarization workflow. The other common container formats, installer packages and disk images, support stapling directly. For example, to staple a ticket to a disk image:

```
% xcrun stapler staple FlyingAnimals.dmg
```

If you don’t staple the ticket to your distribution file, Gatekeeper might block a user from installing or using your product while their Mac is offline.

Notarize your product

If you distribute your product directly, notarize the file you intend to distribute to your users. For more information, see Customizing the notarization workflow. Skip the “Export a package for notarization” section because you already have the file that you want to submit.

If you distribute your product using nested containers, only notarize the outermost container. For example, if you have an app inside an installer package on a disk image, sign the app, sign the installer package, and sign the disk image, but only notarize the disk image.

If you use a third-party installer tool, refer to Customizing the notarization workflow.

If you distribute your product using nested containers, only notarize the outermost container. For example, if you have an app inside an installer package on a disk image, sign the app, sign the installer package, and sign the disk image, but only notarize the disk image.

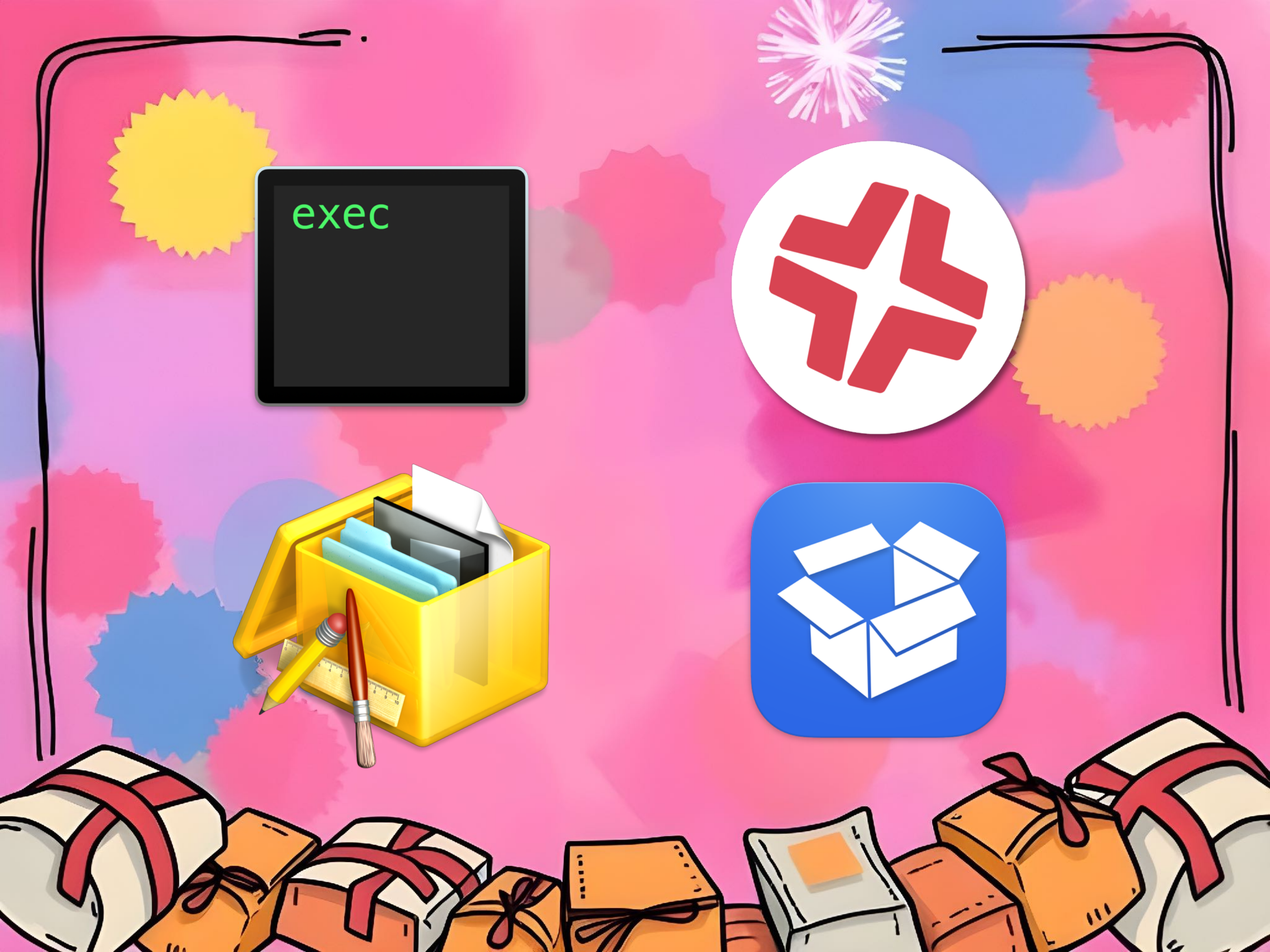
in Customizing the notarization workflow. The other common container formats, installer packages and disk images, support stapling directly. For example, to staple a ticket to a disk image:

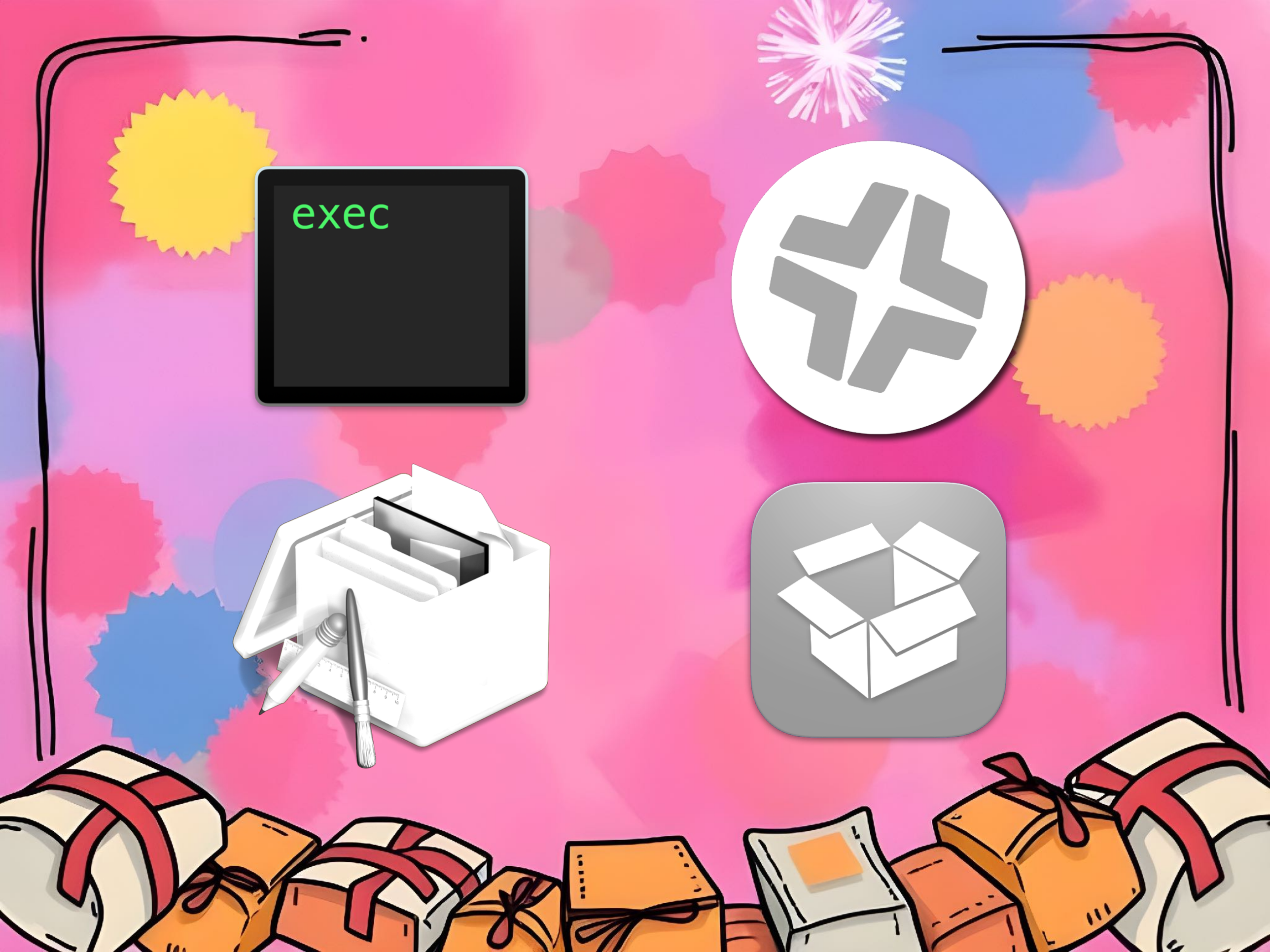
```
% xcrun stapler staple FlyingAnimals.dmg
```

If you don’t staple the ticket to your distribution file, Gatekeeper might block a user from installing or using your product while their Mac is offline.

A flat file that installs a product, that can be signed and notarised, and can run code pre and/or post install of the product.

Inspecting
packages


exec


exec

NAME

pkgutil — Query and manipulate macOS Installer packages and receipts.

SYNOPSIS

pkgutil [options] [commands]

DESCRIPTION

pkgutil reads and manipulates macOS Installer flat packages, and provides access to the "receipt" database used by the Installer. Options are processed first, and affect the operation of all commands. Multiple commands are performed sequentially in the given order.

The files and directories where receipts are stored are subject to change. Always use pkgutil to query or modify them.

OPTIONS

--help, -h
A brief summary of commands and usage.

--force, -f
Don't ask for confirmation before performing a potentially destructive or ambiguous operation.

--verbose, -v
Output in a "human-readable" format with extra headers, footers, indentation, and other contextual information.

--volume path
Perform all operations on the specified volume or home directory. The root volume '/' will be used if unspecified.

--edit-pkg package-id
Specifies an existing receipt to be modified in-place by **--learn**.

--only-files

pkgutil(1) General Commands Manual pkgutil(1)

NAME

pkgutil — Query and manipulate macOS Installer packages and receipts.

SYNOPSIS

pkgutil [options] [commands]

DESCRIPTION

pkgutil reads and manipulates macOS Installer flat packages, and provides access to the "receipt" database used by the Installer. Options are processed first, and affect the operation of all commands. Multiple commands are performed sequentially in the given order.

The files and directories where receipts are stored are subject to change. Always use pkgutil to query or modify them.

DESCRIPTION

pkgutil reads and manipulates macOS Installer flat packages, and provides access to the "receipt" database used by the Installer.

--verbose, -v

Output in a "human-readable" format with extra headers, footers, indentation, and other contextual information.

--volume path

Perform all operations on the specified volume or home directory. The root volume '/' will be used if unspecified.

--edit-pkg package-id

Specifies an existing receipt to be modified in-place by **--learn**.

--only-files

Show the metadata known about path.

--file-info-plist path
Show the metadata known about path in Mac OS X plist(5) format.

FILE COMMANDS

--expand pkg-path dir-path
Expand the flat package at pkg-path into a new directory specified by dir-path.

--flatten dir-path pkg-path
Flatten the dir-path into a new flat package created at pkg-path. The directory to be flattened **must** have the proper contents and layout for a flat package. This is not intended as a substitute for pkgbuild(1).

--bom path
Extract any BOM files from the flat pkg at path into /tmp and return the filename(s). Suggested use is as an argument to lsbom(8). Eg, "lsbom `pkgutil --bom path`". Note that some

FILE COMMANDS

--expand pkg-path dir-path
Expand the flat package at pkg-path into a new directory specified by dir-path.

package(s) is unknown to this command.

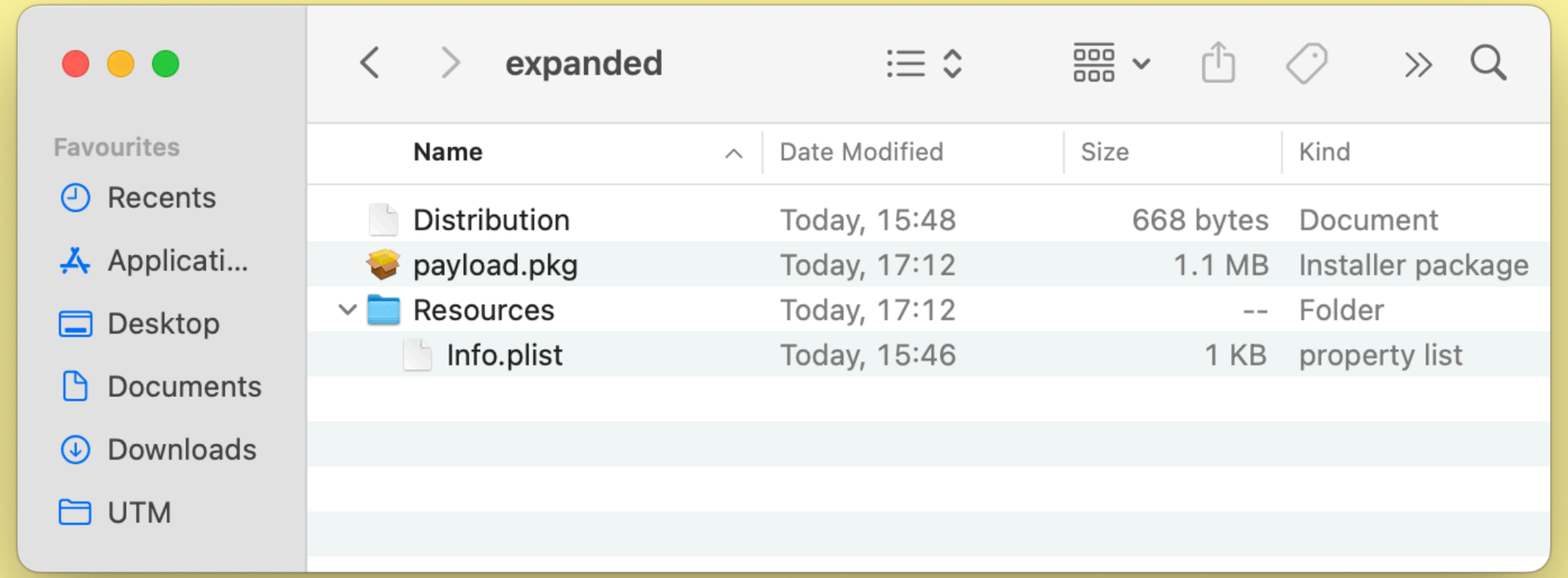
--check-signature pkg-path
Check the validity and trust of the signature on the package at pkg-path. In addition to the status of the signature, the associated certificate chain will be shown.

SEE ALSO

installer(8)
pkgbuild(1)
productbuild(1)

Mac OS X March 2, 2011 Mac OS X
(END)

```
/usr/sbin/pkgutil --expand
~/Desktop/ExampleApp.pkg
~/Desktop/expanded/
```


expanded
Favourites
Recents
Applicati...
Desktop
Documents
Downloads
UTM
Name
Date Modified
Size
Kind
Distribution
Today, 15:48
668 bytes
Document
payload.pkg
Today, 17:12
1.1 MB
Installer package
Resources
Today, 17:12
--
Folder
Info.plist
Today, 15:46
1 KB
property list

Distribution

~/Downloads/UTM/Distribution (no function selected)

```xml
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<installer-script minSpecVersion="1.000000" authoringTool="com.jamfsoftware.Composer" authoringToolVersion="8.4" authoringToolBuild="1">
  <title>Example App</title>
  <options customize="never" allow-external-scripts="no" hostArchitectures="x86_64,arm64"/>
  <domains enable_anywhere="true"/>
  <choices-outline>
    <line choice="payload"/>
  </choices-outline>
  <choice id="payload" title="Example App">
    <pkg-ref id="com.macmule.ExampleApp.payload.pkg"/>
  </choice>
  <pkg-ref id="com.macmule.ExampleApp.payload.pkg" installKBytes="176" version="1" auth="Root">#payload.pkg</pkg-ref>
</installer-script>
```

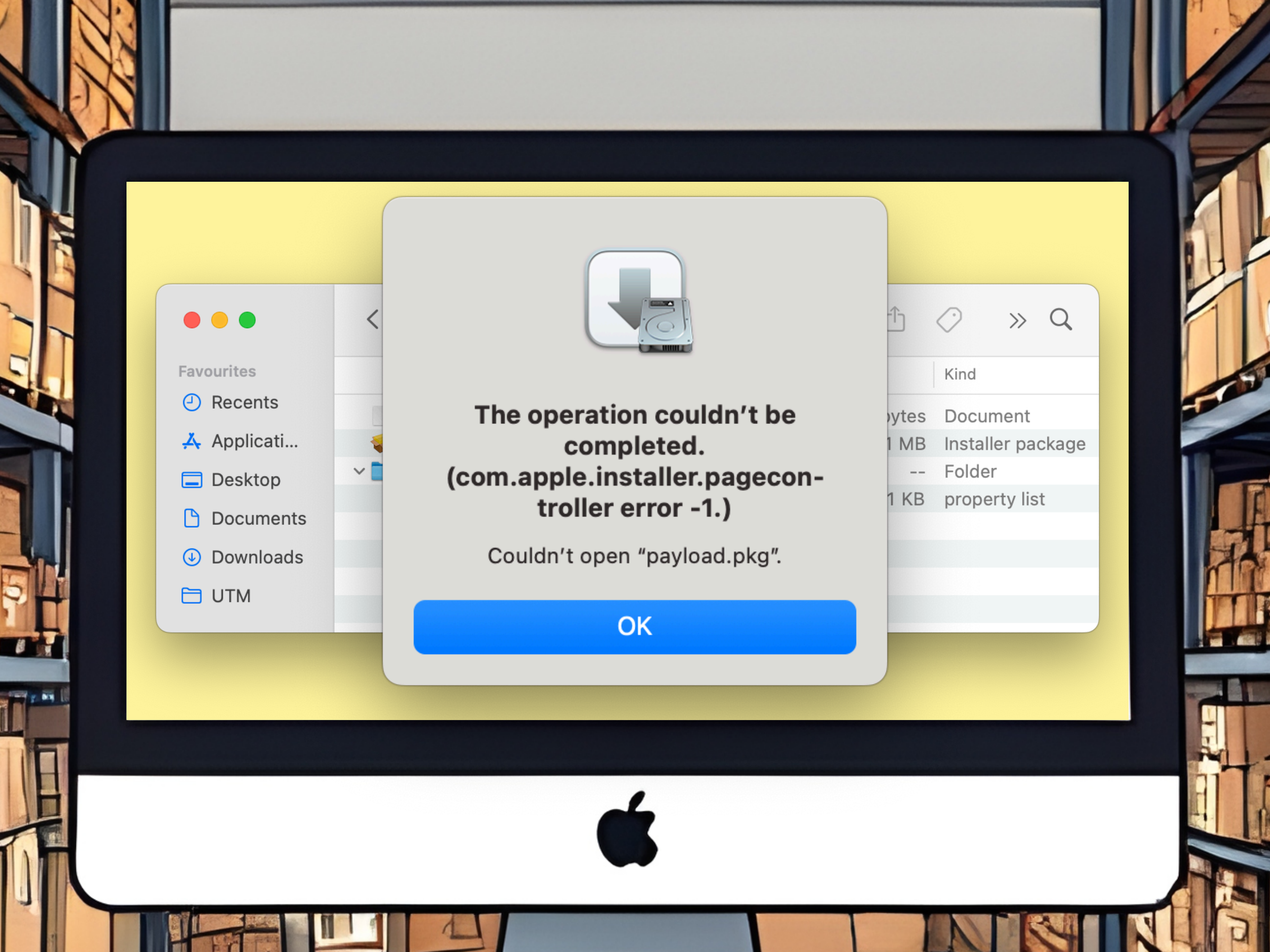
L: 4 C: 86 XML Unicode (UTF-8) Unix (LF) Saved: 17:40:06 668 / 68 / 14 - 100%

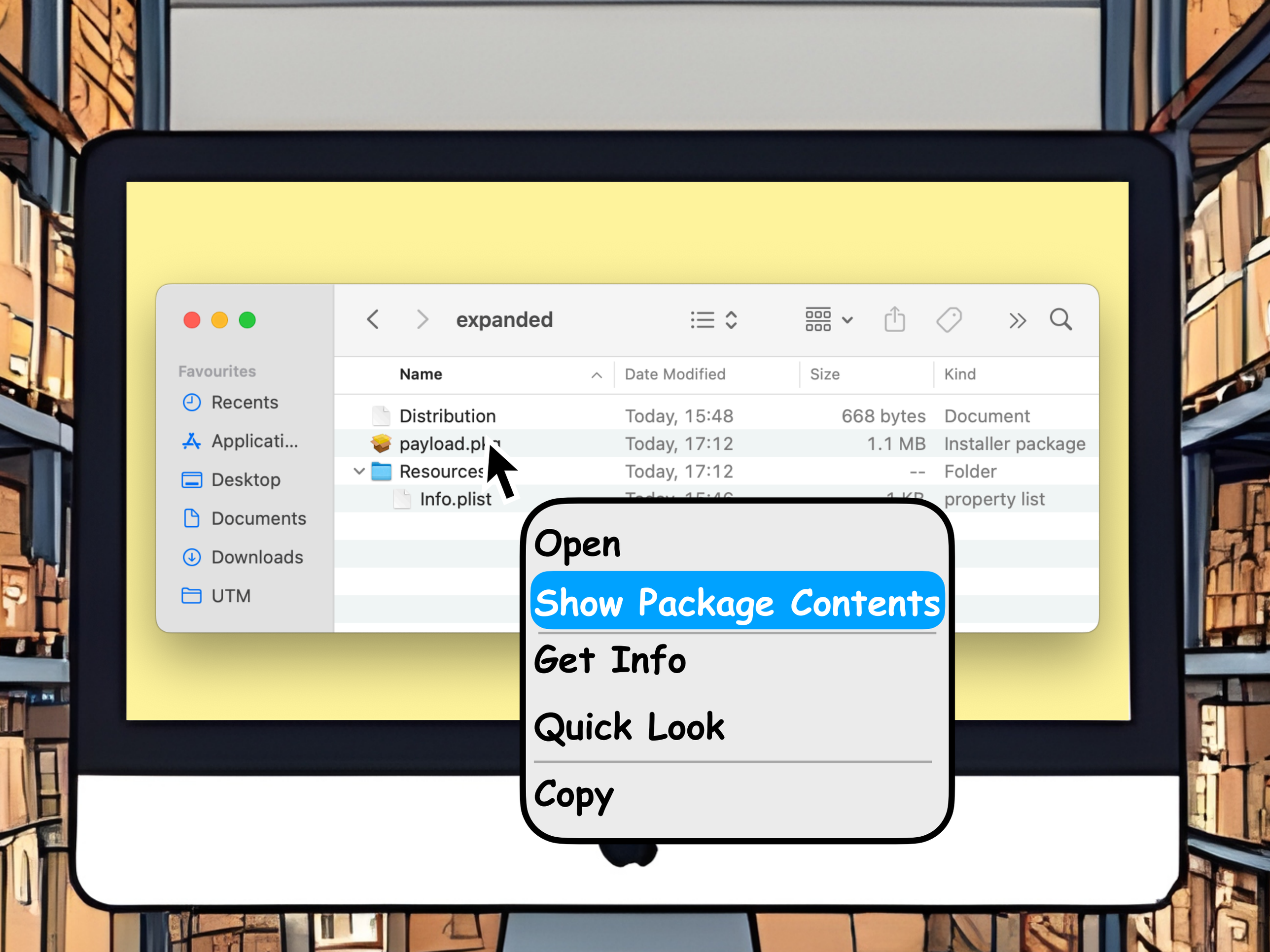
Info.plist

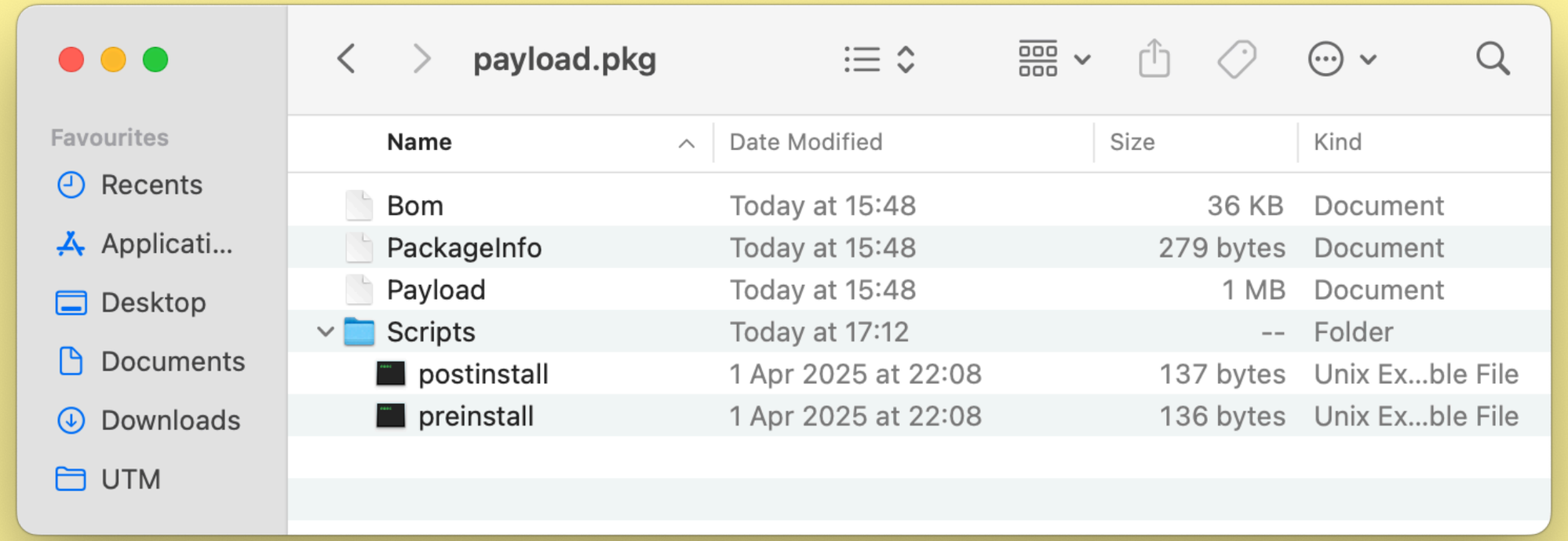
~/Downloads/UTM/Info.plist (functions)


```xml
<?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>CFBundleGetInfoString</key>
    <string>Example App</string>
    <key>CFBundleIdentifier</key>
    <string>com.macmule.ExampleApp</string>
    <key>CFBundleName</key>
    <string>Example App</string>
    <key>CFBundleShortVersionString</key>
    <string>1</string>
    <key>CFBundleVersion</key>
    <string>1.0</string>
    <key>IFMajorVersion</key>
    <string>1</string>
    <key>IFMinorVersion</key>
    <string>0</string>
    <key>IFPkgFlagAllowBackRev</key>
    <false/>
    <key>IFPkgFlagAuthorizationAction</key>
    <string>RootAuthorization</string>
    <key>IFPkgFlagDefaultLocation</key>
    <string>/</string>
    <key>IFPkgFlagFollowLinks</key>
    <true/>
    <key>IFPkgFlagInstallFat</key>
    <false/>
    <key>IFPkgFlagInstalledSize</key>
    <integer>176</integer>
    <key>IFPkgFlagIsRequired</key>
    <false/>
    <key>IFPkgFlagOverwritePermissions</key>
    <false/>
    <key>IFPkgFlagRelocatable</key>
    <false/>
    <key>IFPkgFlagRestartAction</key>
    <string>None</string>
    <key>IFPkgFlagRootVolumeOnly</key>
    <false/>
    <key>IFPkgFlagUpdateInstalledLanguages</key>
    <false/>
    <key>IFPkgFormatVersion</key>
    <string>0.1000000014901161</string>
</dict>
```


L: 46 C: 1 XML Unicode (UTF-8) Unix (LF) Saved: 17:41:36 1,259 / 129 / 46 - 100%


Favourites
Recents
Applicati...
Desktop
Documents
Downloads
UTM
Kind
ytes Document
1 MB Installer package
-- Folder
1 KB property list
The operation couldn't be completed. (com.apple.installer.pagecon-troller error -1.)
Couldn't open "payload.pkg".
OK


expanded
Favourites
Recents
Applicati...
Desktop
Documents
Downloads
UTM
Name
Date Modified
Size
Kind
Distribution
Today, 15:48
668 bytes
Document
payload.pl
Today, 17:12
1.1 MB
Installer package
Resources
Today, 17:12
--
Folder
Info.plist
property list
Open
Show Package Contents
Get Info
Quick Look
Copy


payload.pkg
Favourites
Recents
Applicati...
Desktop
Documents
Downloads
UTM
Name
Date Modified
Size
Kind
Bom
Today at 15:48
36 KB
Document
PackageInfo
Today at 15:48
279 bytes
Document
Payload
Today at 15:48
1 MB
Document
Scripts
Today at 17:12
--
Folder
postinstall
1 Apr 2025 at 22:08
137 bytes
Unix Ex...ble File
preinstall
1 Apr 2025 at 22:08
136 bytes
Unix Ex...ble File

PackageInfo

~/Downloads/PackageInfo (functions)

```
<pkg-info format-version="2" identifier="com.macmule.ExampleApp" version="1" install-location="/" auth="root">
  <payload installKBytes="176" numberOfFiles="17"/>
  <scripts>
    <preinstall file="./preinstall"/>
    <postinstall file="./postinstall"/>
  </scripts>
</pkg-info>
```

L: 1 C: 1 XML Unicode (UTF-8) Unix (LF) Saved: 18:08:58 278 / 28 / 8 - 100%

Bom

~/Downloads/UTM/expanded/payload.pkg/Bom

```
BOMStore¿¿¿¿¿¿@¿¿7z¿¿Uê¿¿#"¿¿¿<¿¿¿¿¿¿
```

NAME

lsbom — list contents of a bom file

SYNOPSIS

lsbom [**-b**] [**-c**] [**-d**] [**-f**] [**-l**] [**-m**] [**-s**] [**-x**] [**--arch** archVal] [**-p** parameters] bom ...
lsbom -h | **--help**

DESCRIPTION

The **lsbom** command interprets the contents of binary bom (bom(5)) files. For each file in a bom, **lsbom** prints the file path and/or requested information.

If no options are given, **lsbom** will display the output formatted such that each line contains the path of the entry, its mode (octal), and its UID/GID. There are slight differences in the output for plain files, directories, symbolic links, and device files as follows:

plain files the UID/GID is followed by the file size and a 32-bit CRC checksum of the file's contents.

symbolic links the UID/GID is followed by the size and checksum of the link path, and the link path itself.

device files the UID/GID file number is followed by the device number.

The **-p** option can be used to specify a user-defined format for **lsbom**'s output. The format string consists of one or more characters described below where each character represents a data type. Data types will be separated by tab characters, and each line will end with a newline character. One can use this mechanism to create output similar to the ls(1) command.

The options are:

-h print full usage

-b list block devices

:

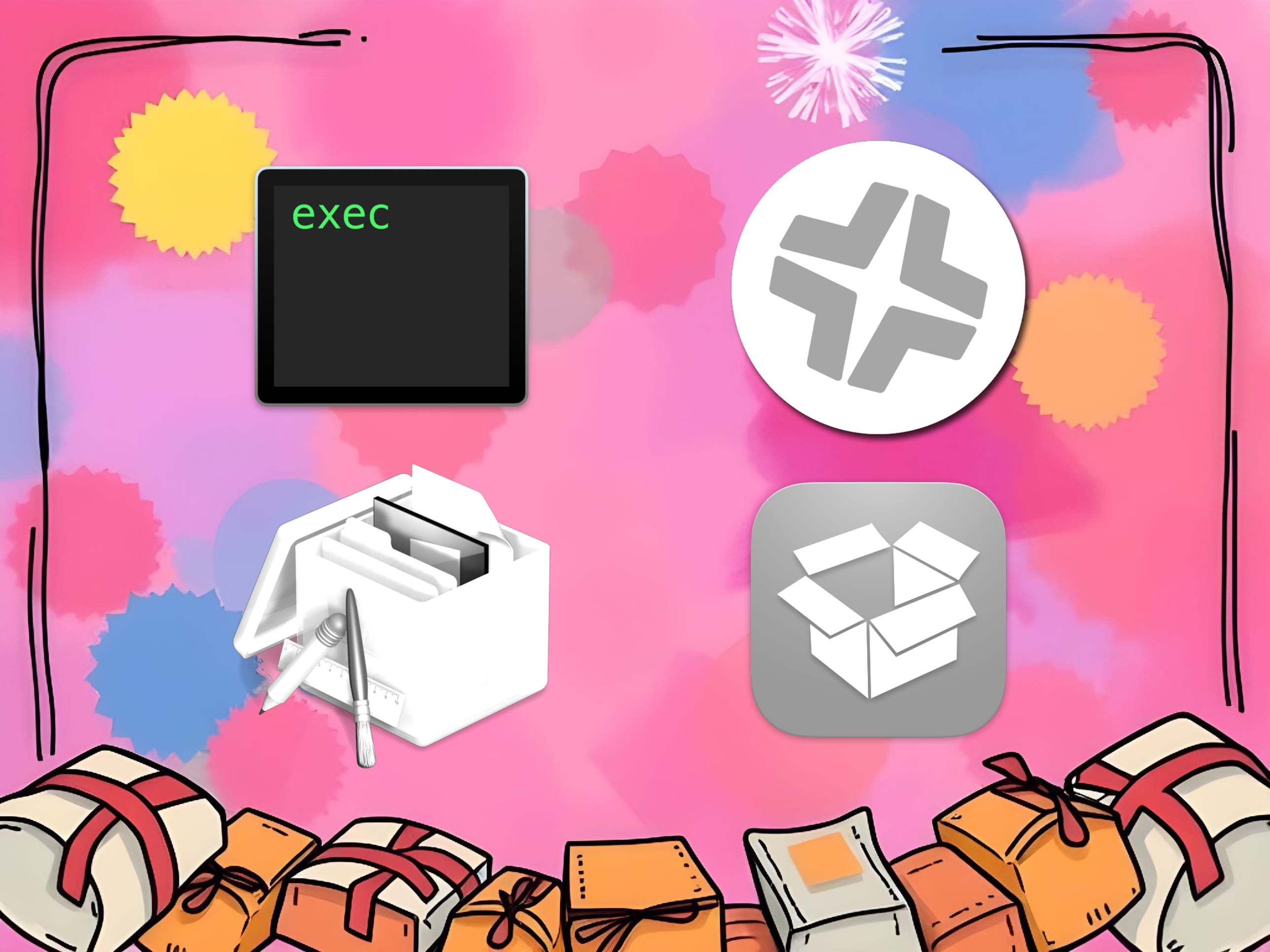
```
/usr/bin/lsbom -s
~/Desktop/expanded/payload.pkg/Bom
```

```
.
./Applications
./Applications/Example App.app
./Applications/Example App.app/Contents
./Applications/Example App.app/Contents/Info.plist
./Applications/Example App.app/Contents/MacOS
./Applications/Example App.app/Contents/MacOS/applet
./Applications/Example App.app/Contents/PkgInfo
./Applications/Example App.app/Contents/Resources
./Applications/Example App.app/Contents/Resources/Scripts
./Applications/Example App.app/Contents/Resources/Scripts/main.scpt
./Applications/Example App.app/Contents/Resources/applet.icns
./Applications/Example App.app/Contents/Resources/applet.rsrc
./Applications/Example App.app/Contents/Resources/description.rtfd
./Applications/Example App.app/Contents/Resources/description.rtfd/TXT.rtf
./Applications/Example App.app/Contents/_CodeSignature
./Applications/Example App.app/Contents/_CodeSignature/CodeResources
./Applications/Example App.app/Icon
```

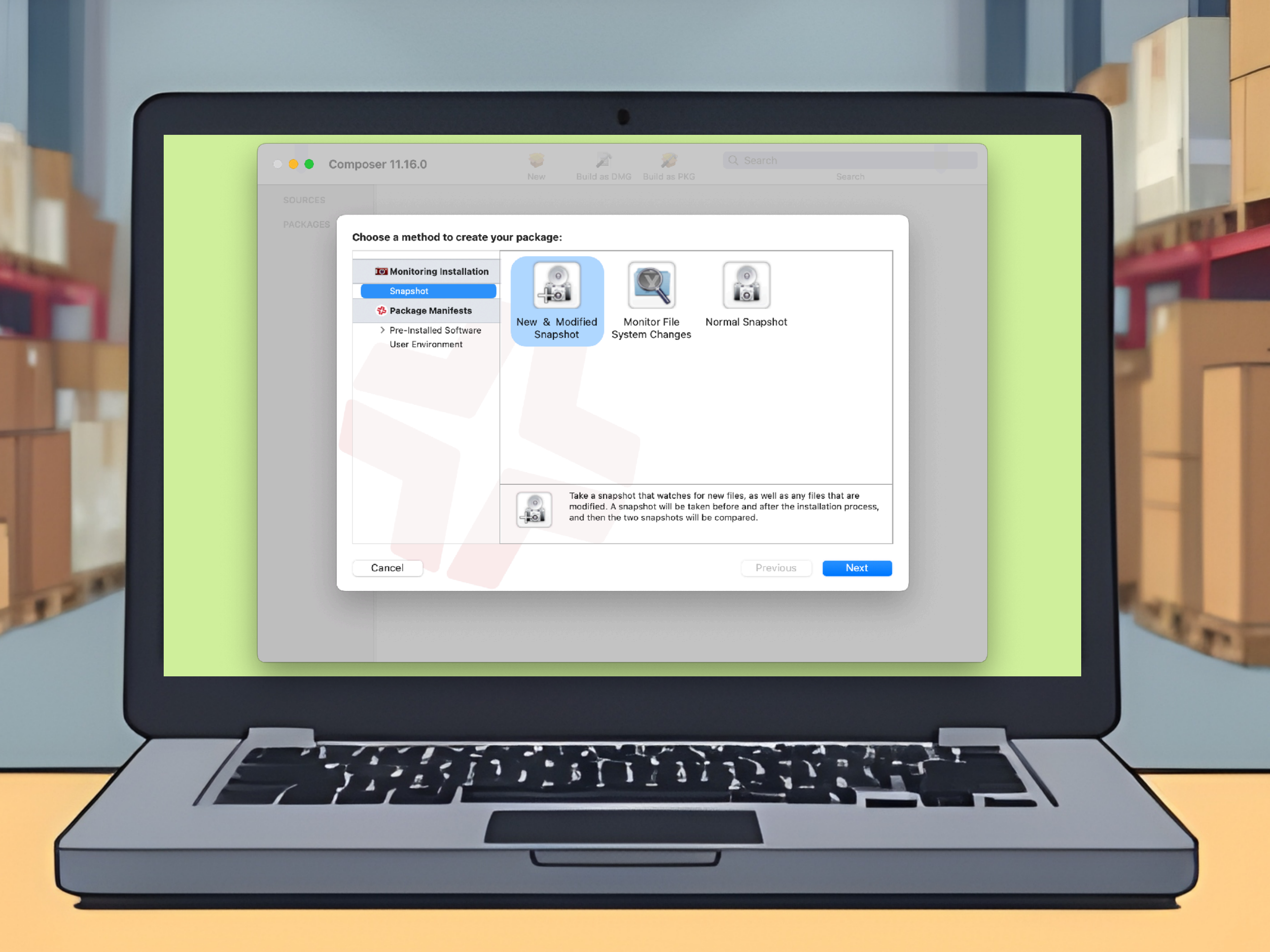
```
TAR(1)                                General Commands Manual                                TAR(1)

NAME
     tar — manipulate tape archives

SYNOPSIS
     tar [bundled-flags <args>] [<file> | <pattern> ...]
     tar {-c} [options] [files | directories]
     tar {-r | -u} -f archive-file [options] [files | directories]
     tar {-t | -x} [options] [patterns]

DESCRIPTION
     tar creates and manipulates streaming archive files.  This implementation can extract from tar, pax, cpio, zip, jar, ar,
     xar, rpm, 7-zip, and ISO 9660 cdrom images and can create tar, pax, cpio, ar, zip, 7-zip, and shar archives.

     The first synopsis form shows a "bundled" option word.  This usage is provided for compatibility with historical
     implementations.  See COMPATIBILITY below for details.

     The other synopsis forms show the preferred usage.  The first option to tar is a mode indicator from the following list:
     -c      Create a new archive containing the specified items.  The long option form is --create.
     -r      Like -c, but new entries are appended to the archive.  Note that this only works on uncompressed archives stored in
             regular files.  The -f option is required.  The long option form is --append.
     -t      List archive contents to stdout.  The long option form is --list.
     -u      Like -r, but new entries are added only if they have a modification date newer than the corresponding entry in the
             archive.  Note that this only works on uncompressed archives stored in regular files.  The -f option is required.
             The long form is --update.
     -x      Extract to disk from the archive.  If a file with the same name appears more than once in the archive, each copy
             will be extracted, with later copies overwriting (replacing) earlier copies.  The long option form is --extract.

     In -c, -r, or -u mode, each specified file or directory is added to the archive in the order specified on the command line.
     By default, the contents of each directory are also archived.

:
```

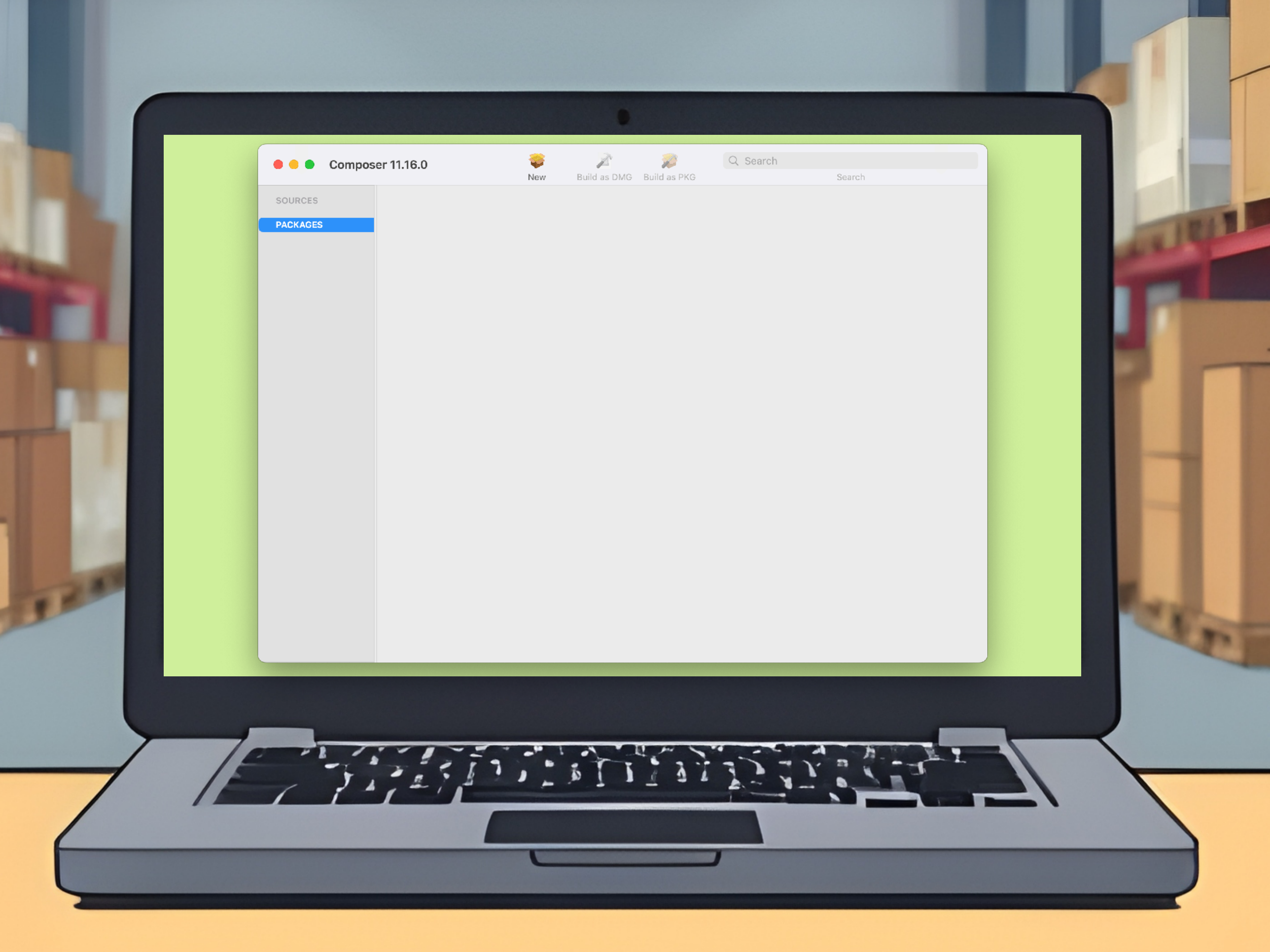
```
/usr/bin/tar xvf
~/Desktop/expanded/payload.pkg/Payload
```

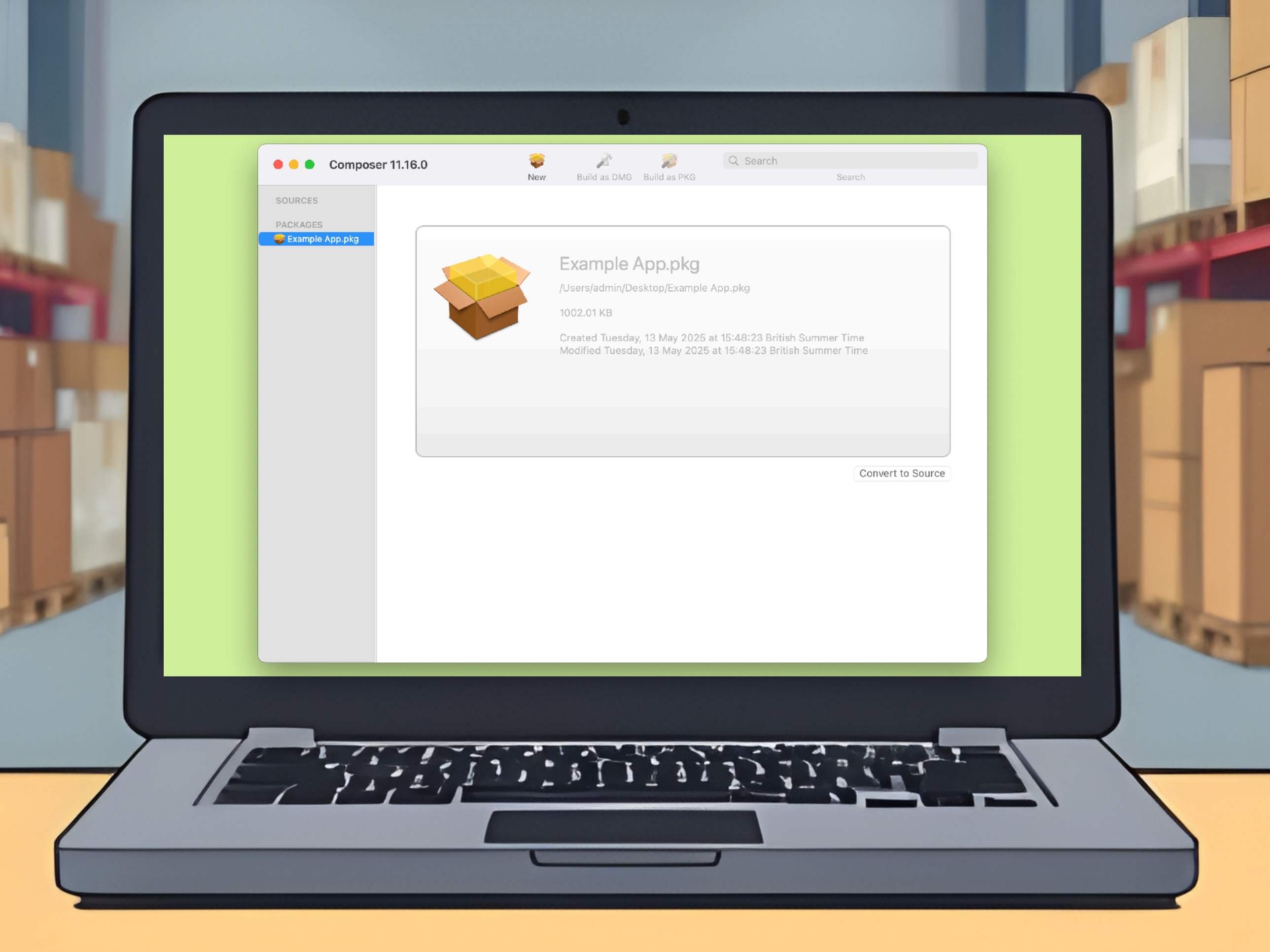
```
x .
x ./Applications
x ./Applications/Example App.app
x ./Applications/Example App.app/Contents
x ./Applications/Example App.app/Contents/_CodeSignature
x ./Applications/Example App.app/Contents/_CodeSignature/CodeResources
x ./Applications/Example App.app/Contents/_CodeSignature/._CodeResources
x ./Applications/Example App.app/Contents/.__CodeSignature
x ./Applications/Example App.app/Contents/MacOS
x ./Applications/Example App.app/Contents/MacOS/applet
x ./Applications/Example App.app/Contents/MacOS/._applet
x ./Applications/Example App.app/Contents/._MacOS
x ./Applications/Example App.app/Contents/Resources
x ./Applications/Example App.app/Contents/Resources/applet.rsrc
x ./Applications/Example App.app/Contents/Resources/._applet.rsrc
x ./Applications/Example App.app/Contents/Resources/description.rtfd
x ./Applications/Example App.app/Contents/Resources/description.rtfd/TXT.rtf
x ./Applications/Example App.app/Contents/Resources/description.rtfd/._TXT.rtf
x ./Applications/Example App.app/Contents/Resources/._description.rtfd
x ./Applications/Example App.app/Contents/Resources/Scripts
x ./Applications/Example App.app/Contents/Resources/Scripts/main.scpt
x ./Applications/Example App.app/Contents/Resources/Scripts/._main.scpt
x ./Applications/Example App.app/Contents/Resources/._Scripts
x ./Applications/Example App.app/Contents/Resources/applet.icns
x ./Applications/Example App.app/Contents/Resources/._applet.icns
x ./Applications/Example App.app/Contents/._Resources
x ./Applications/Example App.app/Contents/Info.plist
x ./Applications/Example App.app/Contents/._Info.plist
x ./Applications/Example App.app/Contents/PkgInfo
x ./Applications/Example App.app/Contents/._PkgInfo
x ./Applications/Example App.app/._Contents
x ./Applications/Example App.app/Icon\r
x ./Applications/Example App.app/._Icon\r
x ./Applications/._Example App.app
x ./._Applications
```

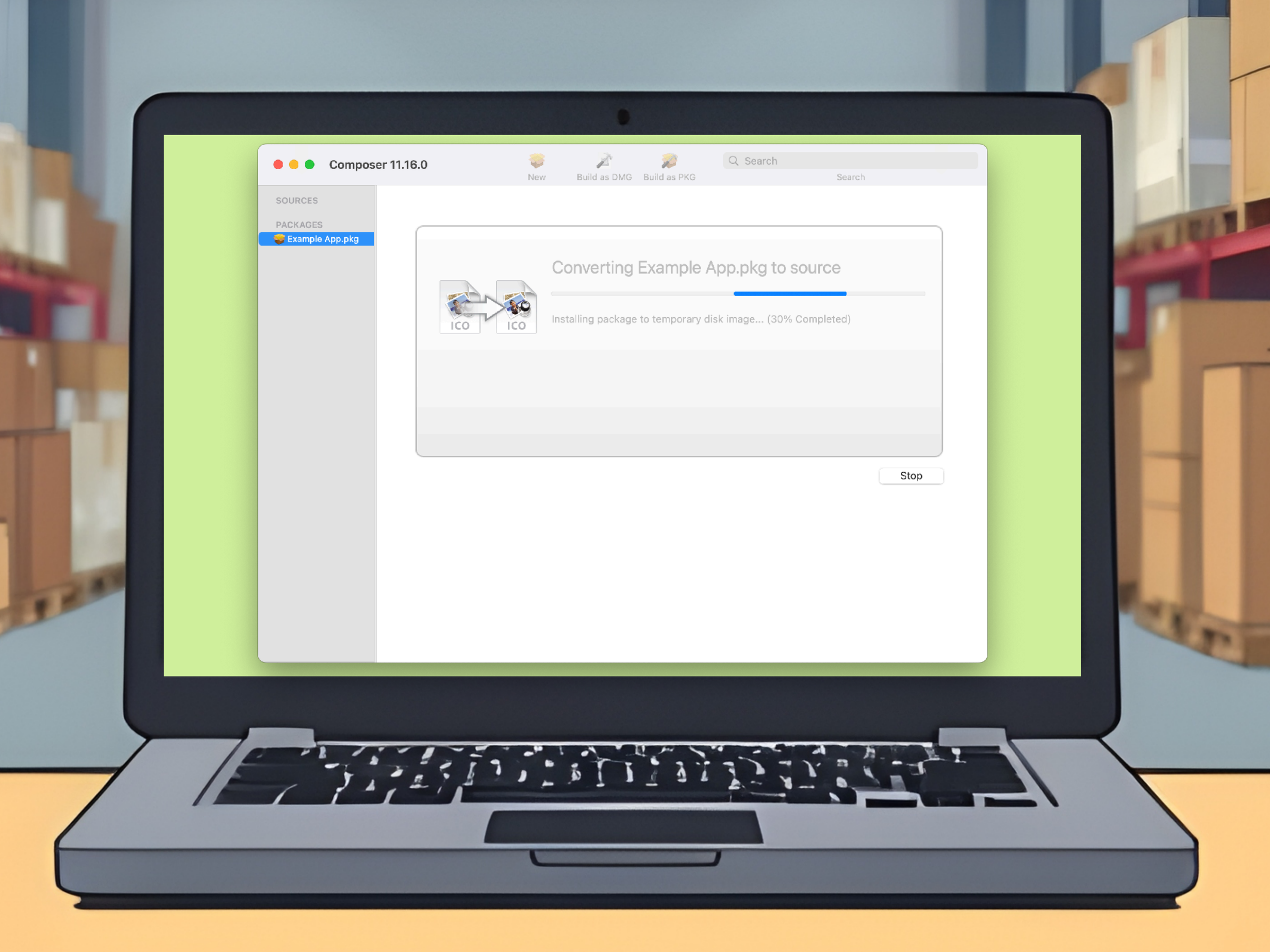

exec

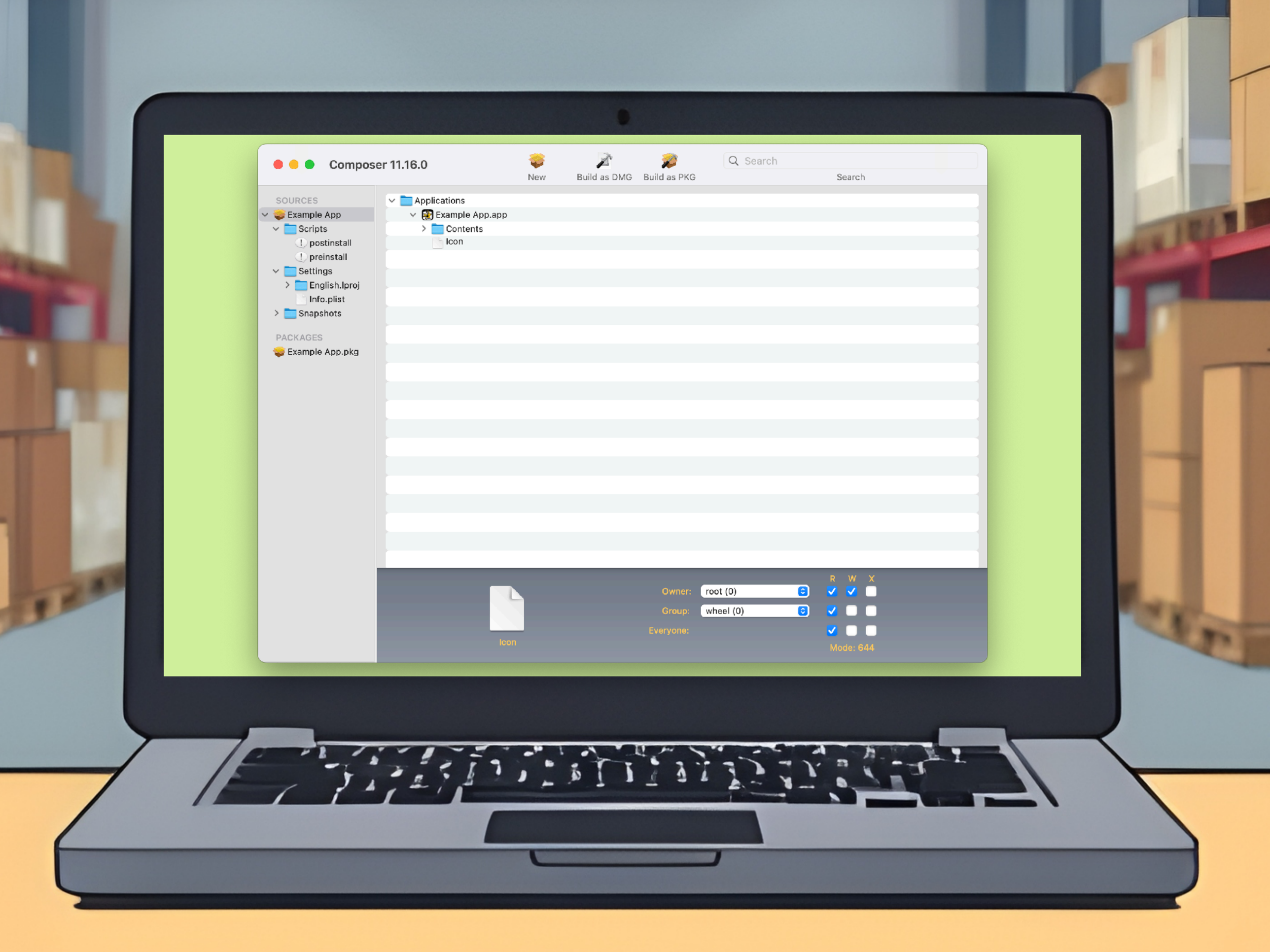
exec


Composer 11.16.0
New
Build as DMG
Build as PKG
Search
Search
SOURCES
PACKAGES
Choose a method to create your package:
Monitoring Installation
Snapshot
Package Manifests
Pre-Installed Software
User Environment
New & Modified Snapshot
Monitor File System Changes
Normal Snapshot
Take a snapshot that watches for new files, as well as any files that are modified. A snapshot will be taken before and after the installation process, and then the two snapshots will be compared.
Cancel
Previous
Next


Composer 11.16.0
New
Build as DMG
Build as PKG
Search
Search
SOURCES
PACKAGES


Composer 11.16.0
New
Build as DMG
Build as PKG
Search
Search
SOURCES
PACKAGES
Example App.pkg
Example App.pkg
/Users/admin/Desktop/Example App.pkg
1002.01 KB
Created Tuesday, 13 May 2025 at 15:48:23 British Summer Time
Modified Tuesday, 13 May 2025 at 15:48:23 British Summer Time
Convert to Source

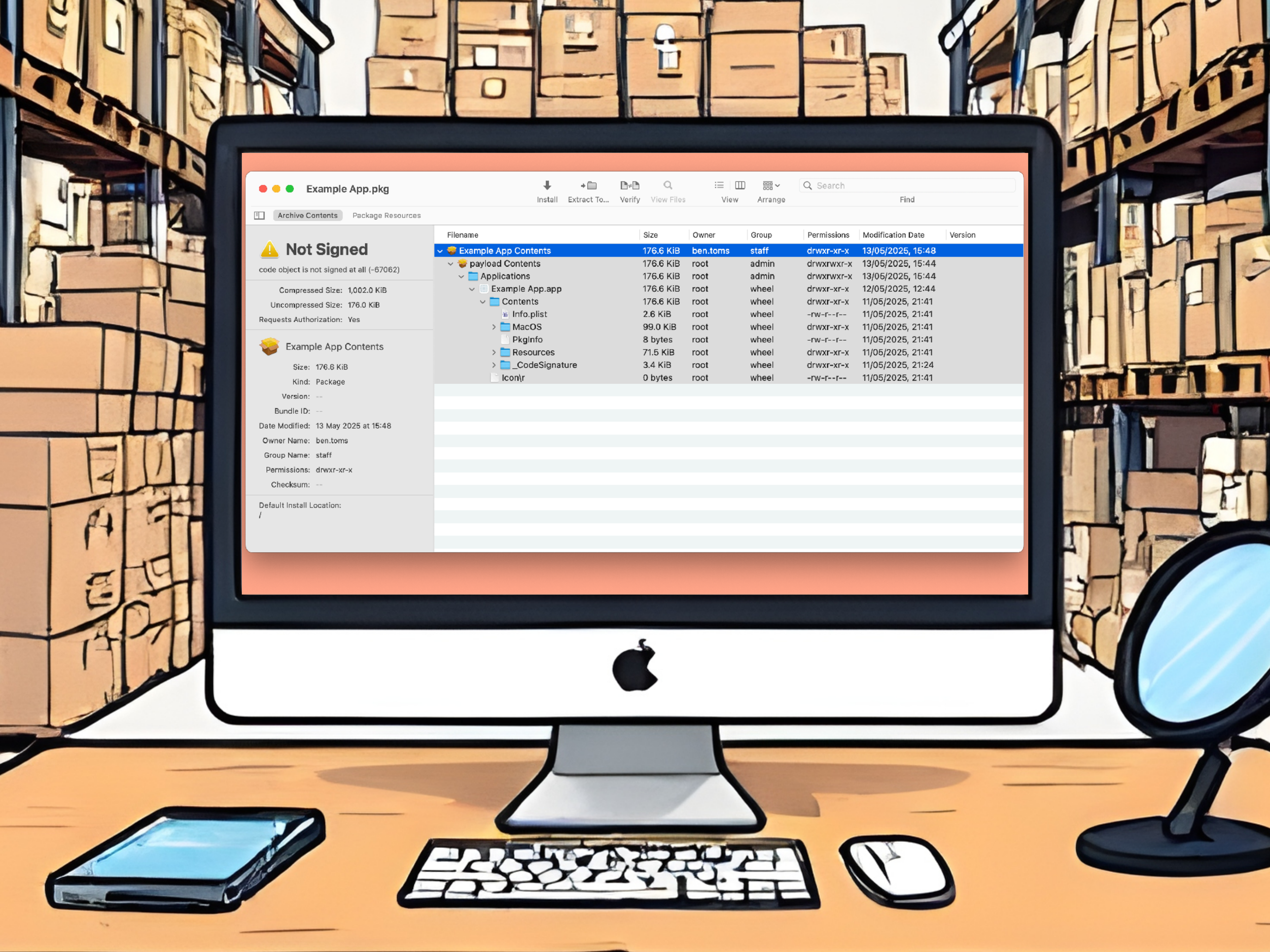

Composer 11.16.0
New
Build as DMG
Build as PKG
Search
Search
SOURCES
PACKAGES
Example App.pkg
Converting Example App.pkg to source
ICO
ICO
Installing package to temporary disk image... (30% Completed)
Stop

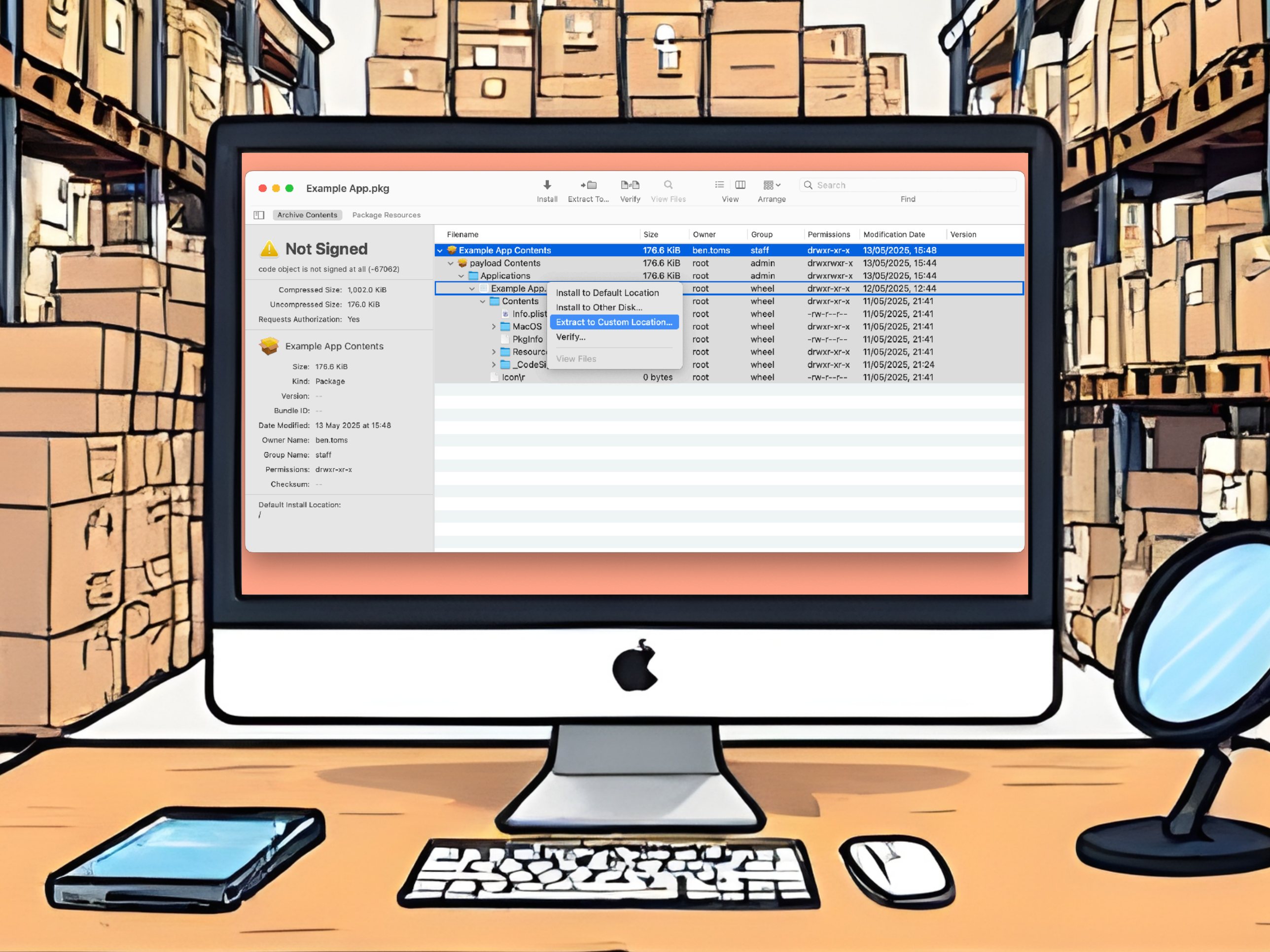

Composer 11.16.0
New
Build as DMG
Build as PKG
Search
Search
SOURCES
Example App
Scripts
postinstall
preinstall
Settings
English.lproj
Info.plist
Snapshots
PACKAGES
Example App.pkg
Applications
Example App.app
Contents
Icon
Icon
Owner:
root (0)
Group:
wheel (0)
Everyone:
R
W
X
Mode: 644

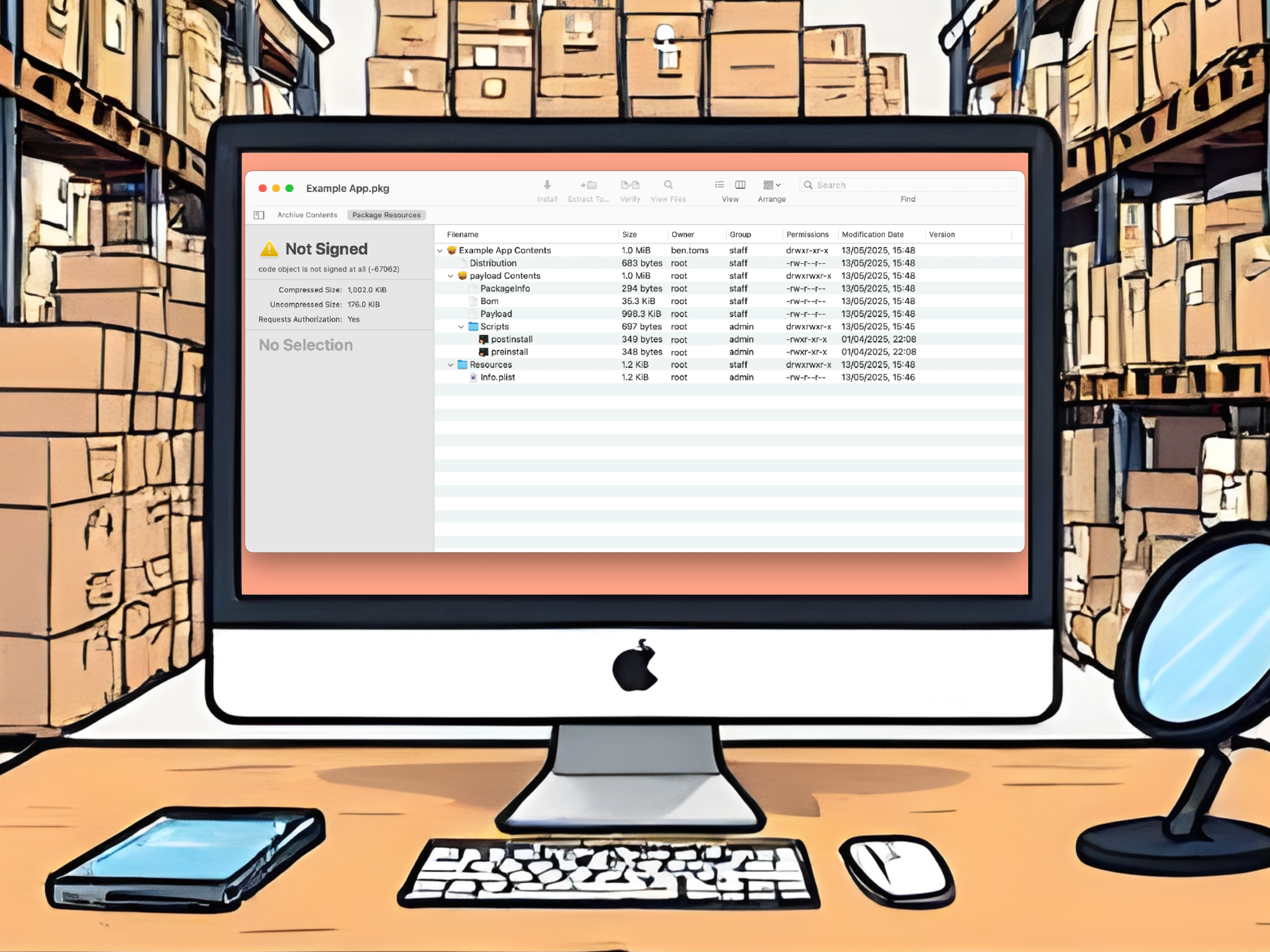
exec

exec

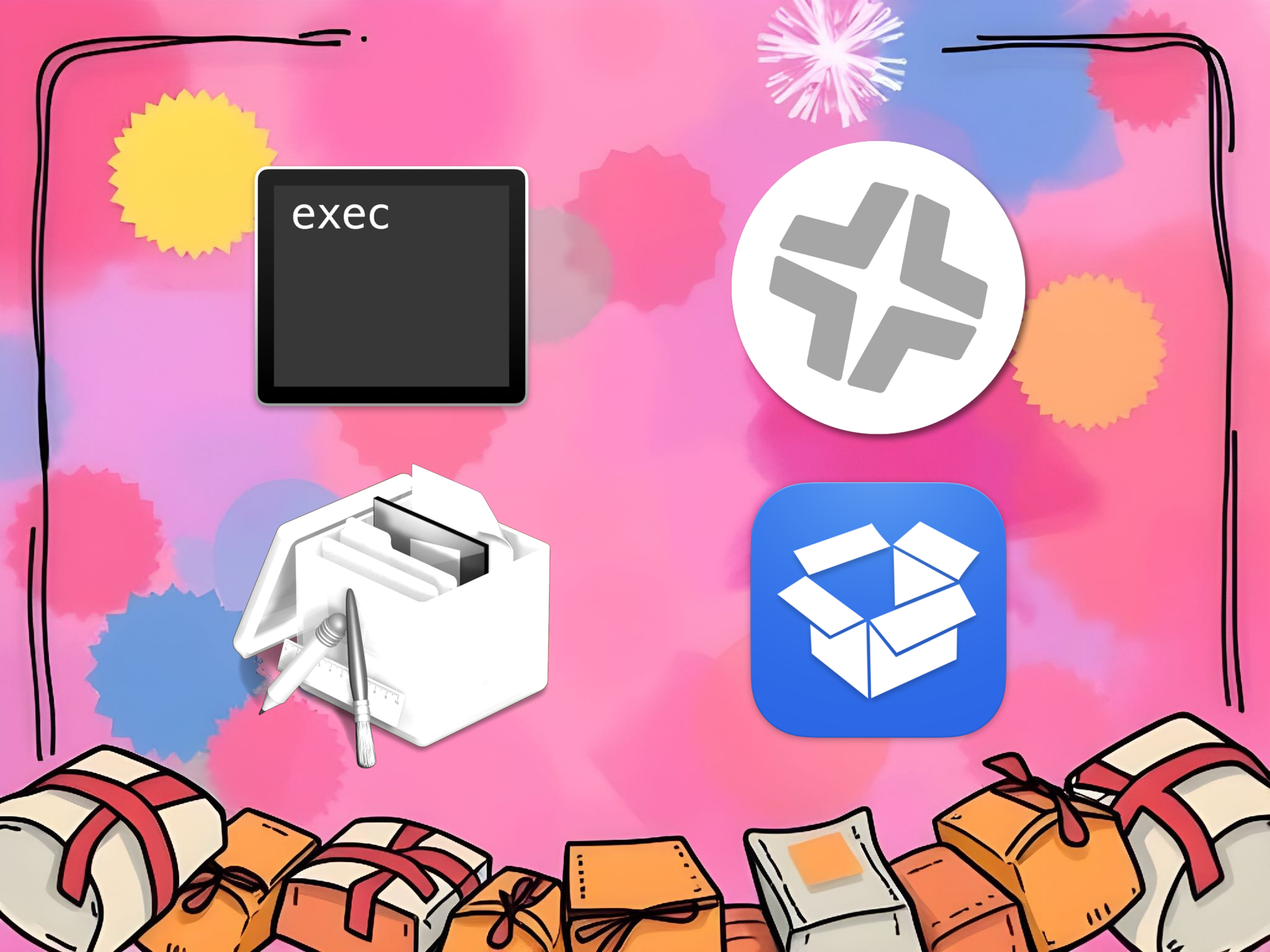
Pacifist
Version 4.1.0
Registered To:
Ben Toms
charlessoft.com
Contact
System Add-Ons:
QuickLook Preview Extension
Built-in
Settings...
Command-Line Tool
Not Installed
Install
Browser Plug-In for Chrome
Not Installed
Install
Browser Plug-In for Firefox
Not Installed
Install
Drag a package, archive, or disk image file onto this window to open it, or choose one of the following options:
Open Package
Open an installer package, .zip, .tar, .cpio, or .xar archive, or disk image
Open URL
Open a package, archive, or disk image remotely over the World Wide Web
Analyze Existing Installations
Find out what files on your Mac have been installed by third-party installers

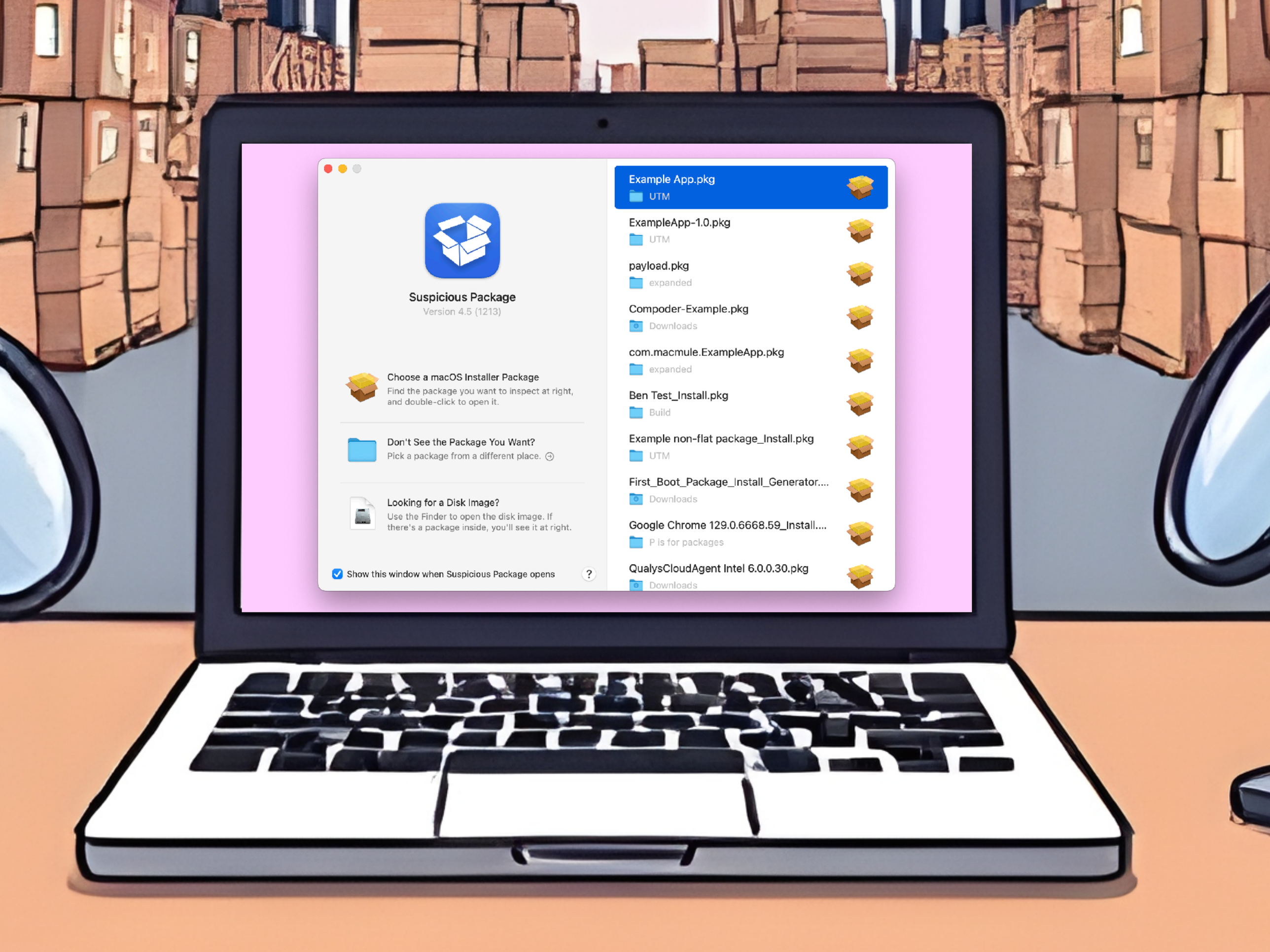

Example App.pkg
Install
Extract To...
Verify
View Files
View
Arrange
Search
Find
Archive Contents
Package Resources
Not Signed
code object is not signed at all (-67062)
Compressed Size: 1,002.0 KiB
Uncompressed Size: 176.0 KiB
Requests Authorization: Yes
Example App Contents
Size: 176.6 KiB
Kind: Package
Version: --
Bundle ID: --
Date Modified: 13 May 2025 at 15:48
Owner Name: ben.toms
Group Name: staff
Permissions: drwxr-xr-x
Checksum: --
Default Install Location:
/
Filename Size Owner Group Permissions Modification Date Version
Example App Contents 176.6 KiB ben.toms staff drwxr-xr-x 13/05/2025, 15:48
payload Contents 176.6 KiB root admin drwxrwxr-x 13/05/2025, 15:44
Applications 176.6 KiB root admin drwxrwxr-x 13/05/2025, 15:44
Example App.app 176.6 KiB root wheel drwxr-xr-x 12/05/2025, 12:44
Contents 176.6 KiB root wheel drwxr-xr-x 11/05/2025, 21:41
Info.plist 2.6 KiB root wheel -rw-r--r-- 11/05/2025, 21:41
MacOS 99.0 KiB root wheel drwxr-xr-x 11/05/2025, 21:41
PkgInfo 8 bytes root wheel -rw-r--r-- 11/05/2025, 21:41
Resources 71.5 KiB root wheel drwxr-xr-x 11/05/2025, 21:41
_CodeSignature 3.4 KiB root wheel drwxr-xr-x 11/05/2025, 21:24
Icon\r 0 bytes root wheel -rw-r--r-- 11/05/2025, 21:41

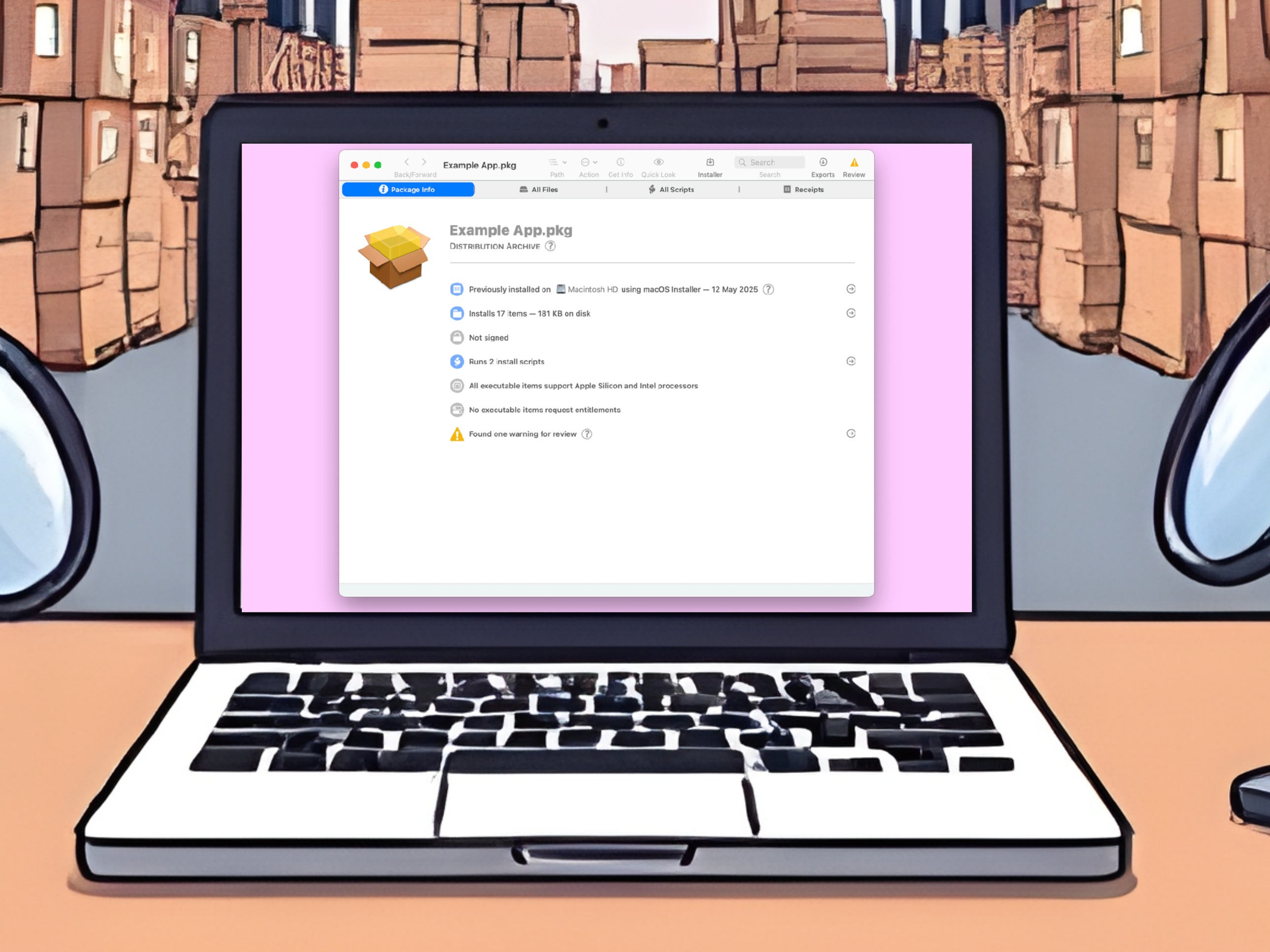

Example App.pkg
Install
Extract To...
Verify
View Files
View
Arrange
Search
Find
Archive Contents
Package Resources
Not Signed
code object is not signed at all (-67062)
Compressed Size: 1,002.0 KiB
Uncompressed Size: 176.0 KiB
Requests Authorization: Yes
Example App Contents
Size: 176.6 KiB
Kind: Package
Version: --
Bundle ID: --
Date Modified: 13 May 2025 at 15:48
Owner Name: ben.toms
Group Name: staff
Permissions: drwxr-xr-x
Checksum: --
Default Install Location:
/
Filename
Size
Owner
Group
Permissions
Modification Date
Version
Example App Contents 176.6 KiB ben.toms staff drwxr-xr-x 13/05/2025, 15:48
payload Contents 176.6 KiB root admin drwxrwxr-x 13/05/2025, 15:44
Applications 176.6 KiB root admin drwxrwxr-x 13/05/2025, 15:44
Example App. root wheel drwxr-xr-x 12/05/2025, 12:44
Contents root wheel drwxr-xr-x 11/05/2025, 21:41
Info.plist root wheel -rw-r--r-- 11/05/2025, 21:41
MacOS root wheel drwxr-xr-x 11/05/2025, 21:41
PkgInfo root wheel -rw-r--r-- 11/05/2025, 21:41
Resourc root wheel drwxr-xr-x 11/05/2025, 21:41
_CodeSi root wheel drwxr-xr-x 11/05/2025, 21:24
Icon\r 0 bytes root wheel -rw-r--r-- 11/05/2025, 21:41
Install to Default Location
Install to Other Disk...
Extract to Custom Location...
Verify...
View Files

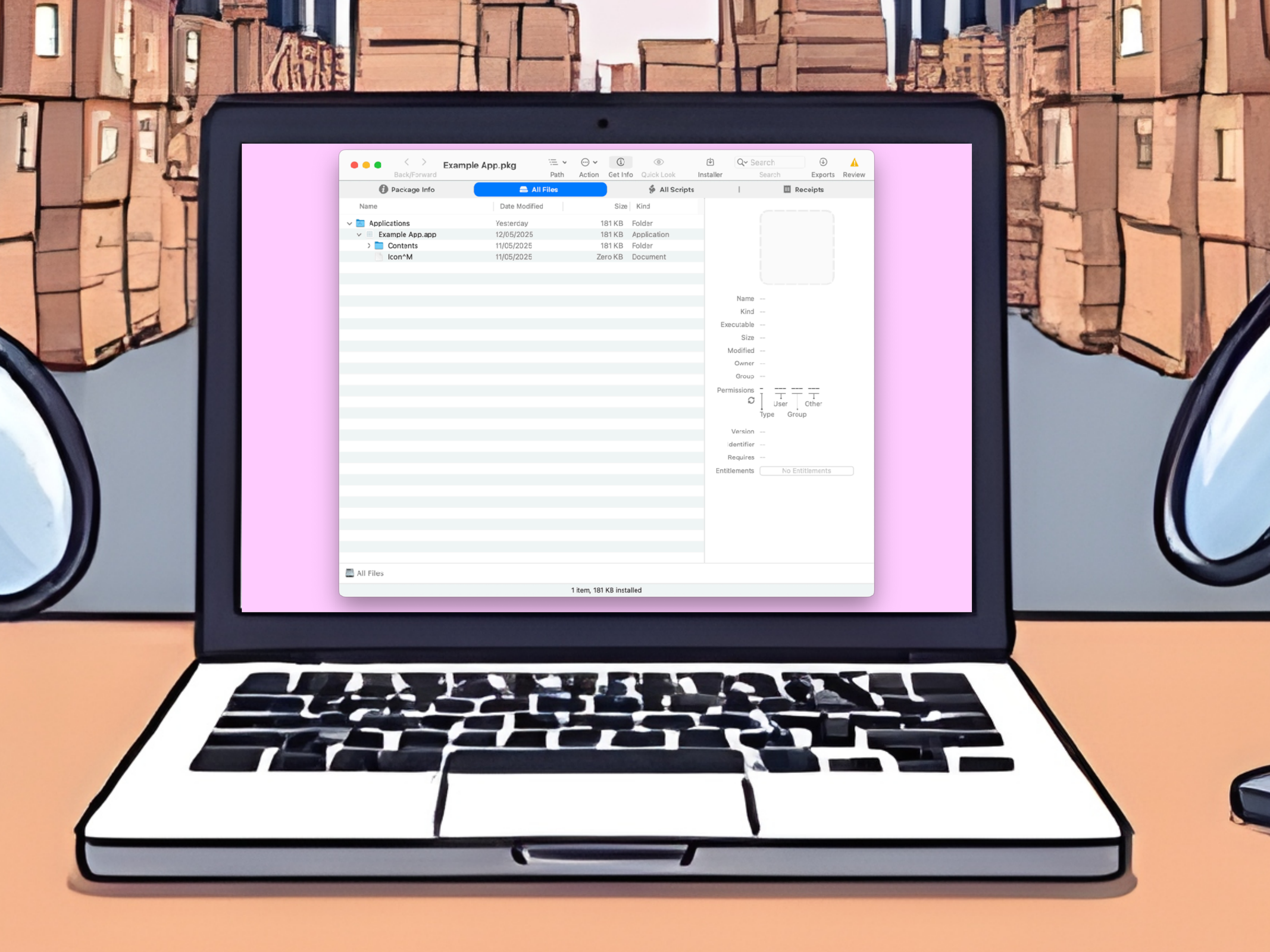

Example App.pkg
Install
Extract To...
Verify
View Files
View
Arrange
Search
Find
Archive Contents
Package Resources
Not Signed
code object is not signed at all (-67062)
Compressed Size: 1,002.0 KiB
Uncompressed Size: 176.0 KiB
Requests Authorization: Yes
No Selection
Filename Size Owner Group Permissions Modification Date Version
Example App Contents 1.0 MiB ben.toms staff drwxr-xr-x 13/05/2025, 15:48
Distribution 683 bytes root staff -rw-r--r-- 13/05/2025, 15:48
payload Contents 1.0 MiB root staff drwxrwxr-x 13/05/2025, 15:48
PackageInfo 294 bytes root staff -rw-r--r-- 13/05/2025, 15:48
Bom 35.3 KiB root staff -rw-r--r-- 13/05/2025, 15:48
Payload 998.3 KiB root staff -rw-r--r-- 13/05/2025, 15:48
Scripts 697 bytes root admin drwxrwxr-x 13/05/2025, 15:45
postinstall 349 bytes root admin -rwxr-xr-x 01/04/2025, 22:08
preinstall 348 bytes root admin -rwxr-xr-x 01/04/2025, 22:08
Resources 1.2 KiB root staff drwxrwxr-x 13/05/2025, 15:48
Info.plist 1.2 KiB root admin -rw-r--r-- 13/05/2025, 15:46

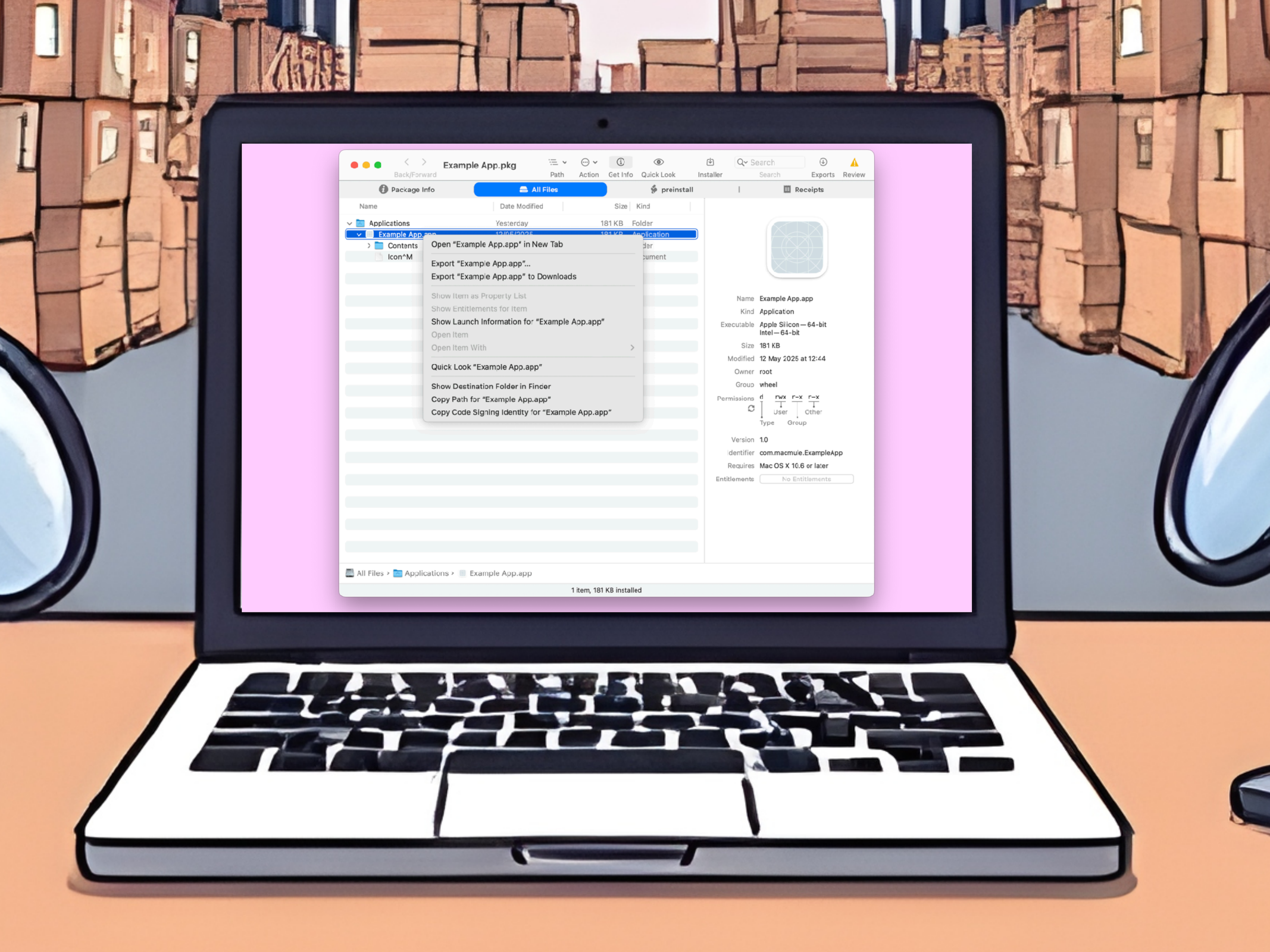
exec

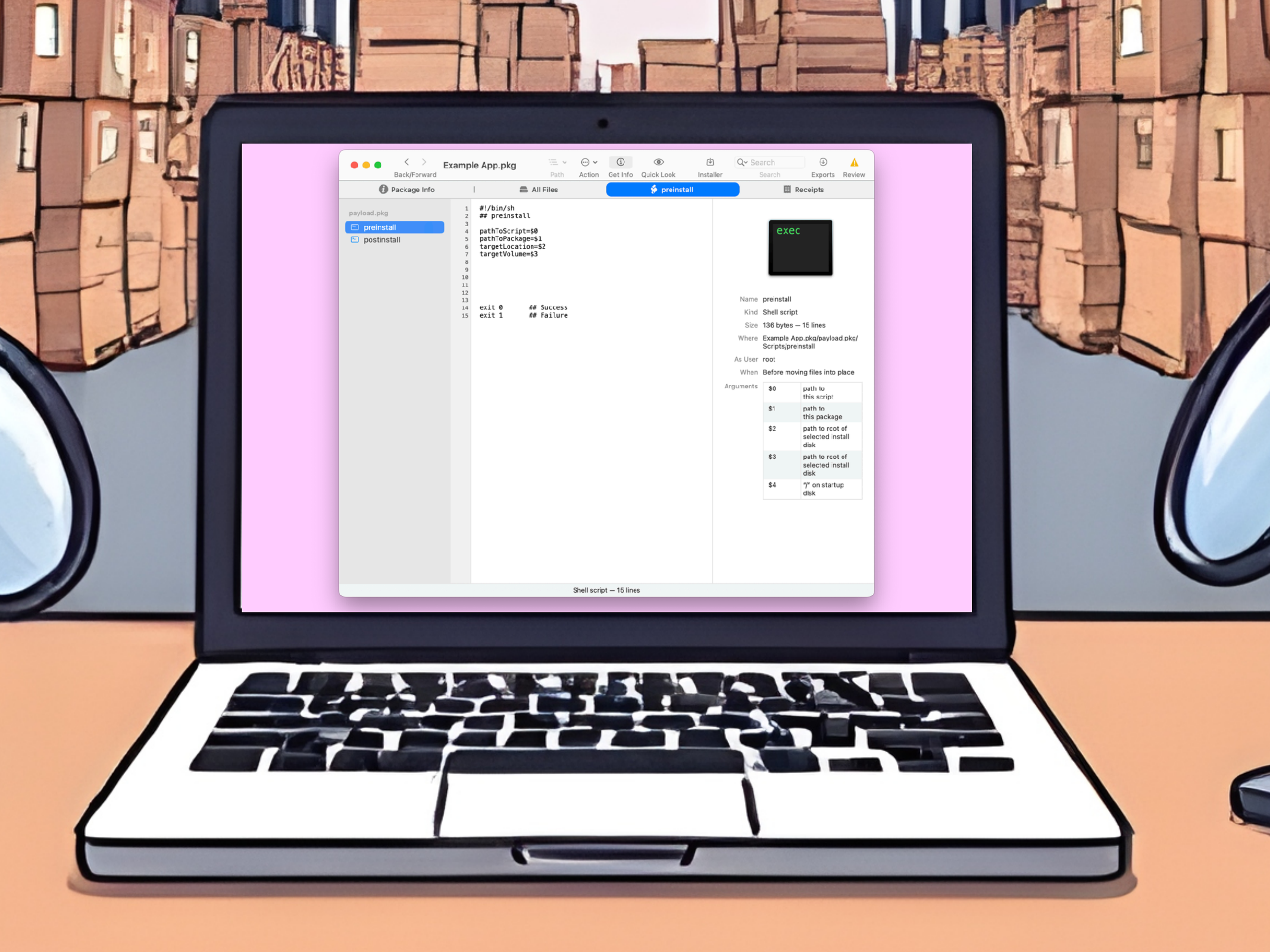

exec

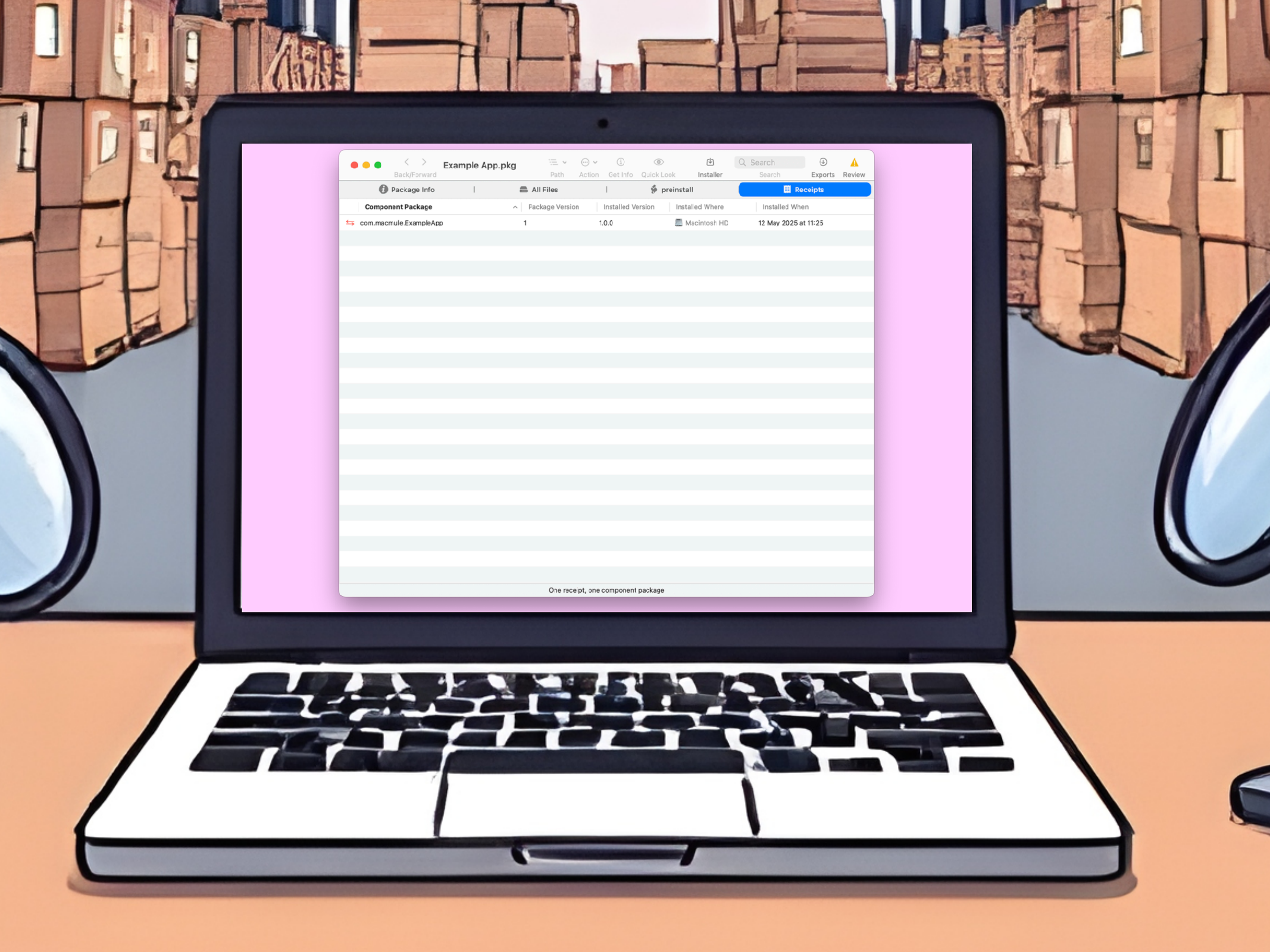

Suspicious Package
Version 4.5 (1213)
Choose a macOS Installer Package
Find the package you want to inspect at right, and double-click to open it.
Don't See the Package You Want?
Pick a package from a different place.
Looking for a Disk Image?
Use the Finder to open the disk image. If there's a package inside, you'll see it at right.
Show this window when Suspicious Package opens
Example App.pkg
UTM
ExampleApp-1.0.pkg
UTM
payload.pkg
expanded
Compoder-Example.pkg
Downloads
com.macmule.ExampleApp.pkg
expanded
Ben Test_Install.pkg
Build
Example non-flat package_Install.pkg
UTM
First_Boot_Package_Install_Generator....
Downloads
Google Chrome 129.0.6668.59_Install....
P is for packages
QualysCloudAgent Intel 6.0.0.30.pkg
Downloads


Example App.pkg
Back/Forward
Path
Action
Get Info
Quick Look
Installer
Search
Exports
Review
Package Info
All Files
All Scripts
Receipts
Example App.pkg
Distribution Archive
Previously installed on Macintosh HD using macOS Installer — 12 May 2025
Installs 17 items — 181 KB on disk
Not signed
Runs 2 install scripts
All executable items support Apple Silicon and Intel processors
No executable items request entitlements
Found one warning for review

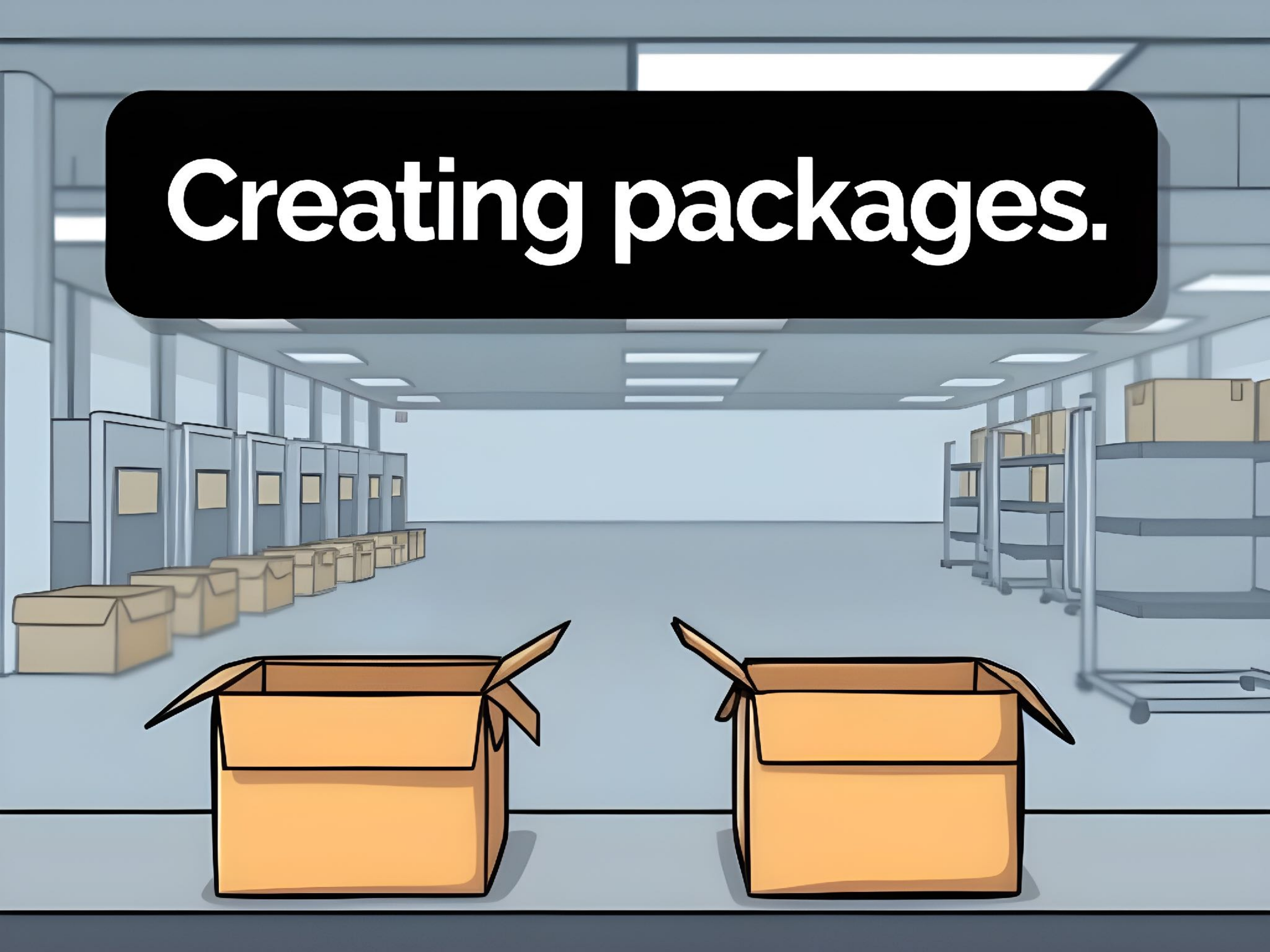

Example App.pkg
Back/Forward
Path
Action
Get Info
Quick Look
Installer
Search
Exports
Review
Package Info
All Files
All Scripts
Receipts
Name
Date Modified
Size
Kind
Applications
Yesterday
181 KB
Folder
Example App.app
12/05/2025
181 KB
Application
Contents
11/05/2025
181 KB
Folder
Icon^M
11/05/2025
Zero KB
Document
Name
Kind
Executable
Size
Modified
Owner
Group
Permissions
User
Other
Type
Group
Version
Identifier
Requires
Entitlements
No Entitlements
All Files
1 item, 181 KB installed

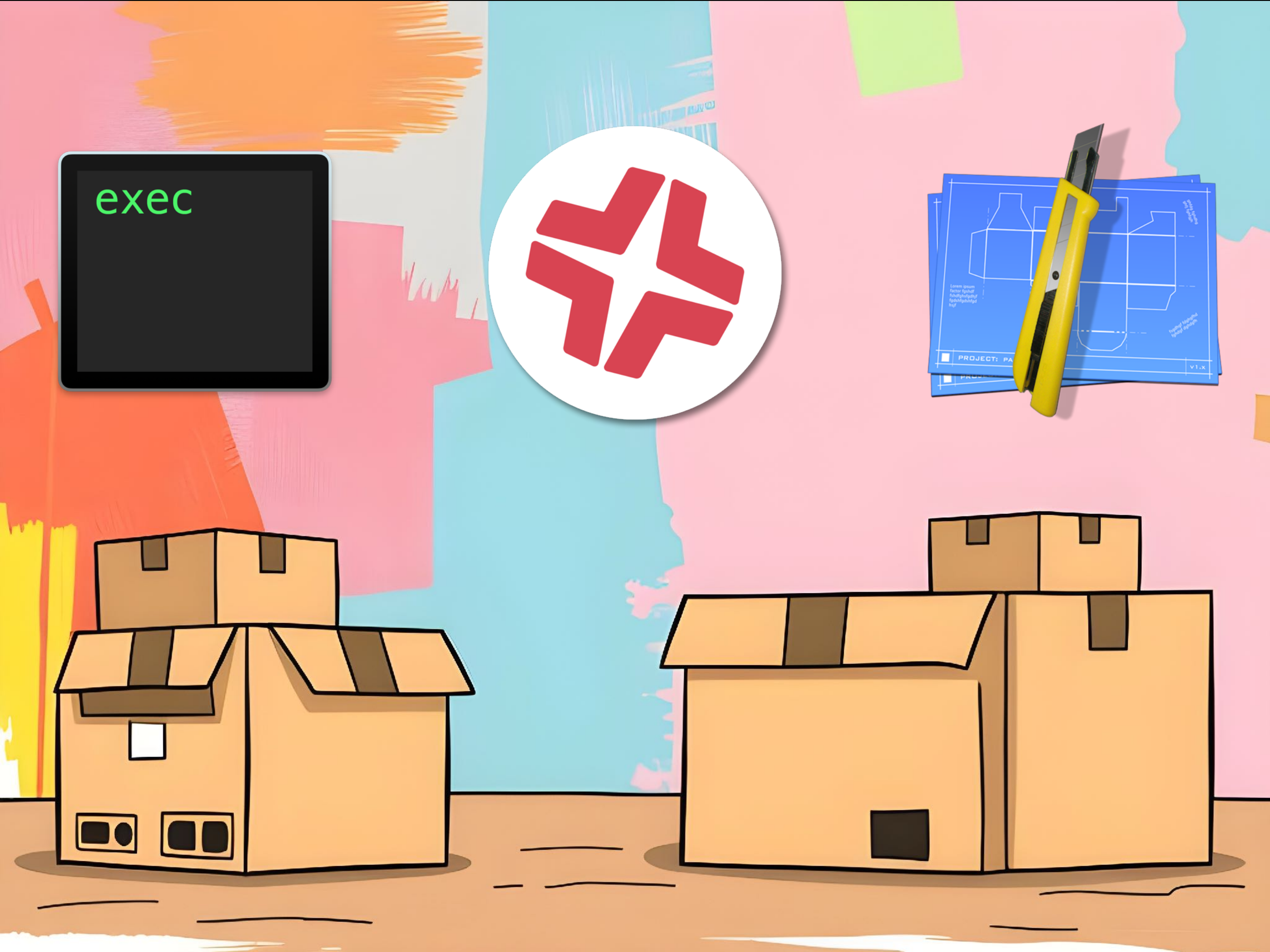

Example App.pkg
Back/Forward
Path
Action
Get Info
Quick Look
Installer
Search
Exports
Review
Package Info
All Files
preinstall
Receipts
Name
Date Modified
Size
Kind
Applications
Yesterday
181 KB
Folder
Example App.app
Application
Contents
Icon^M
Open "Example App.app" in New Tab
Export "Example App.app"...
Export "Example App.app" to Downloads
Show Item as Property List
Show Entitlements for Item
Show Launch Information for "Example App.app"
Open Item
Open Item With
Quick Look "Example App.app"
Show Destination Folder in Finder
Copy Path for "Example App.app"
Copy Code Signing Identity for "Example App.app"
Name Example App.app
Kind Application
Executable Apple Silicon — 64-bit
Intel — 64-bit
Size 181 KB
Modified 12 May 2025 at 12:44
Owner root
Group wheel
Permissions d rwx r-x r-x
User
Other
Type
Group
Version 1.0
Identifier com.macmule.ExampleApp
Requires Mac OS X 10.6 or later
Entitlements No Entitlements
All Files › Applications › Example App.app
1 item, 181 KB installed

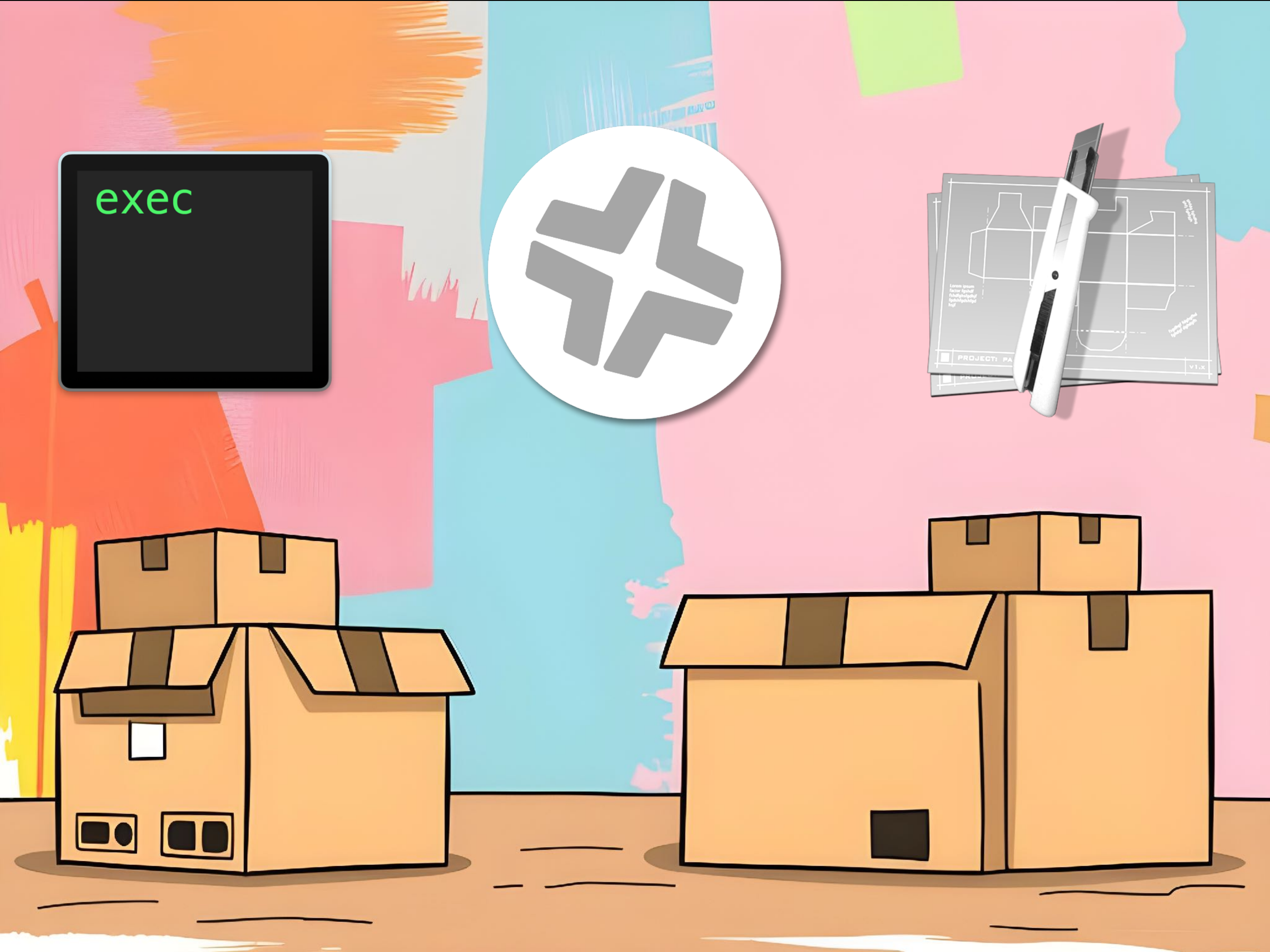

Example App.pkg
Back/Forward
Path
Action
Get Info
Quick Look
Installer
Search
Exports
Review
Package Info
All Files
preinstall
Receipts
payload.pkg
preinstall
postinstall
#!/bin/sh
preinstall
pathToScript=$0
pathToPackage=$1
targetLocation=$2
targetVolume=$3
exit 0 ## Success
exit 1 ## Failure
exec
Name preinstall
Kind Shell script
Size 136 bytes — 15 lines
Where Example App.pkg/payload.pkg/Scripts/preinstall
As User root
When Before moving files into place
Arguments
$0 path to this script
$1 path to this package
$2 path to root of selected install disk
$3 path to root of selected install disk
$4 "/" on startup disk
Shell script — 15 lines

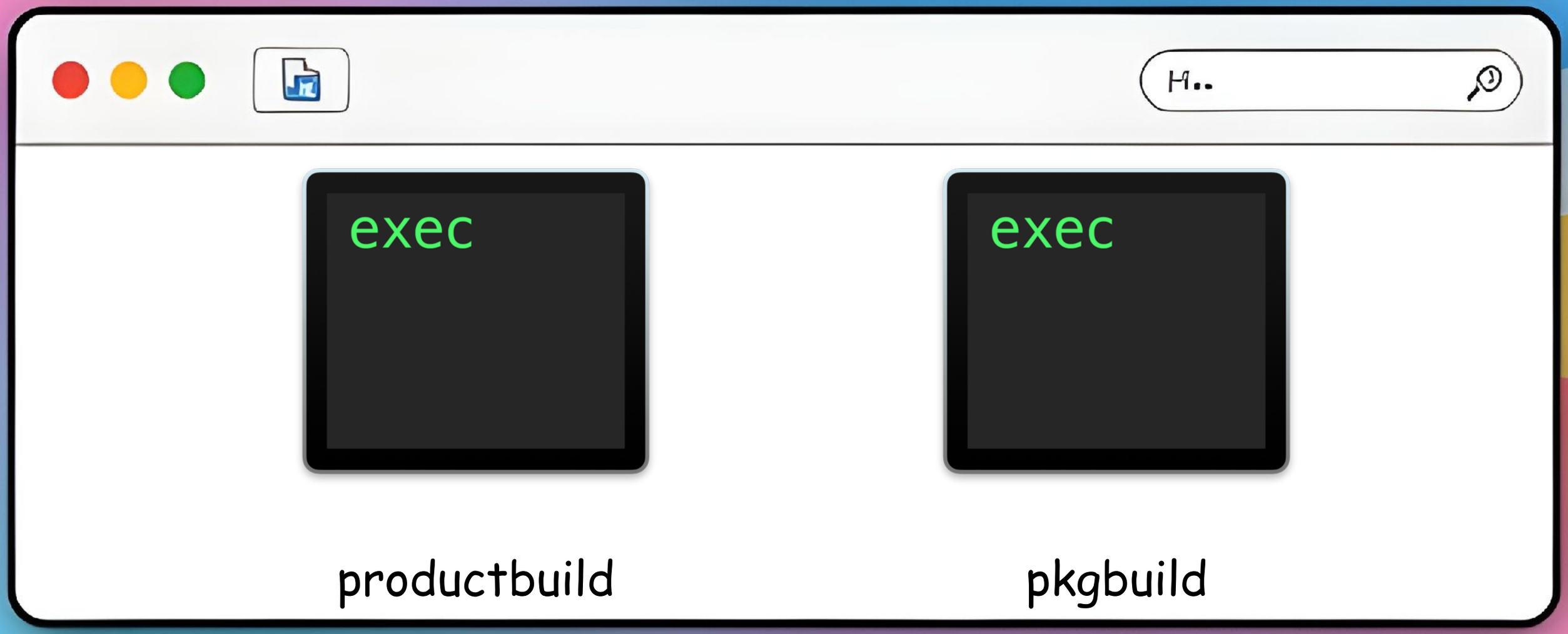

Example App.pkg
Back/Forward
Path
Action
Get Info
Quick Look
Installer
Search
Exports
Review
Package Info
All Files
preinstall
Receipts
Component Package
Package Version
Installed Version
Installed Where
Installed When
com.macmule.ExampleApp
1
1.0.0
Macintosh HD
12 May 2025 at 11:25
One receipt, one component package

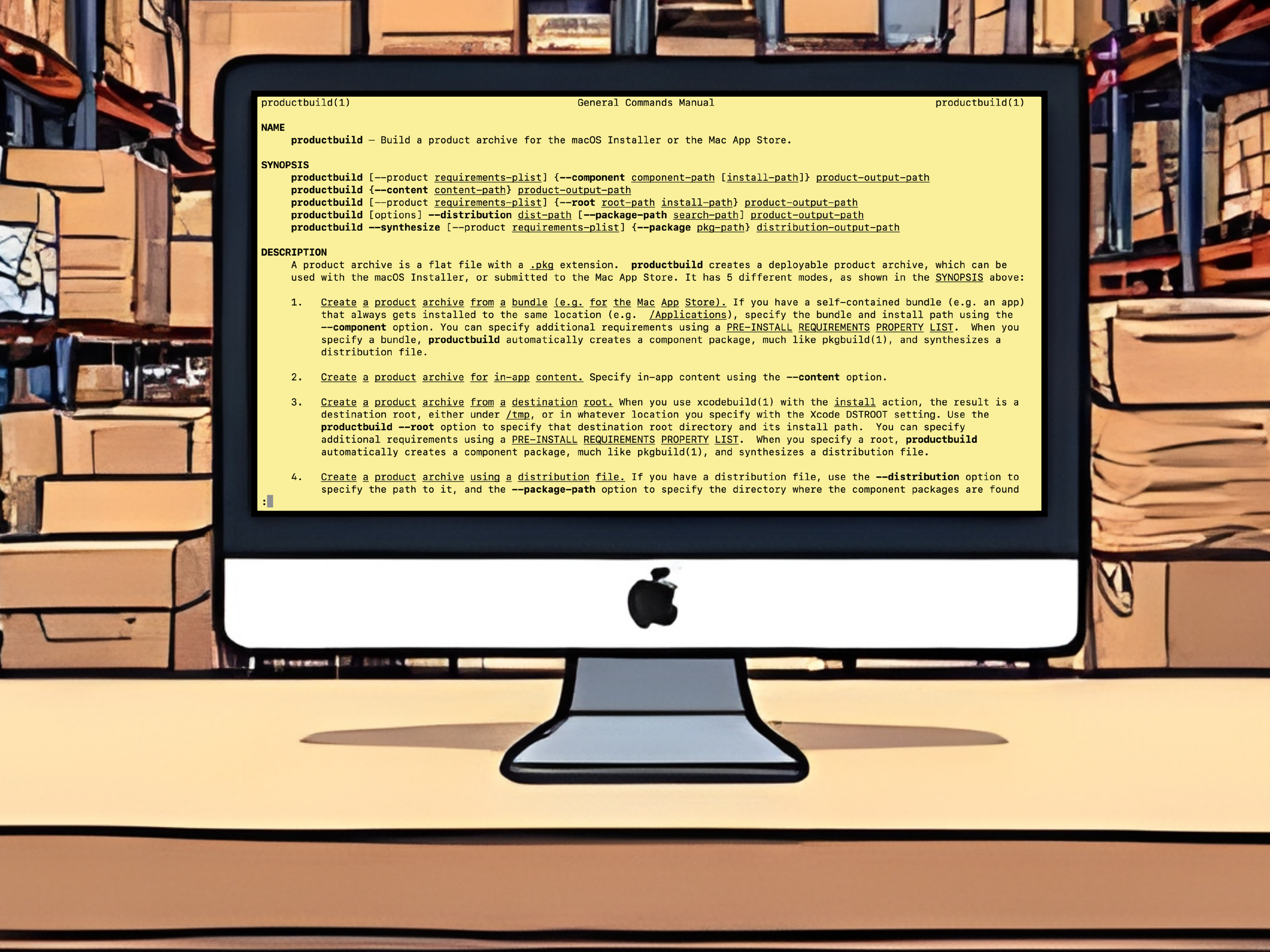
Creating packages.




exec
PROJECT:
v1.x


exec


exec
exec
productbuild
pkgbuild



productbuild(1) General Commands Manual productbuild(1)

NAME

productbuild — Build a product archive for the macOS Installer or the Mac App Store.

SYNOPSIS

productbuild [--product requirements-plist] {**--component** component-path [install-path]} product-output-path
productbuild {**--content** content-path} product-output-path
productbuild [--product requirements-plist] {**--root** root-path install-path} product-output-path
productbuild [options] **--distribution** dist-path [**--package-path** search-path] product-output-path
productbuild **--synthesize** [--product requirements-plist] {**--package** pkg-path} distribution-output-path

DESCRIPTION

A product archive is a flat file with a .pkg extension. **productbuild** creates a deployable product archive, which can be used with the macOS Installer, or submitted to the Mac App Store. It has 5 different modes, as shown in the SYNOPSIS above:

1. Create a product archive from a bundle (e.g. for the Mac App Store). If you have a self-contained bundle (e.g. an app) that always gets installed to the same location (e.g. /Applications), specify the bundle and install path using the **--component** option. You can specify additional requirements using a PRE-INSTALL REQUIREMENTS PROPERTY LIST. When you specify a bundle, **productbuild** automatically creates a component package, much like pkgbuild(1), and synthesizes a distribution file.

2. Create a product archive for in-app content. Specify in-app content using the **--content** option.

3. Create a product archive from a destination root. When you use xcodebuild(1) with the install action, the result is a destination root, either under /tmp, or in whatever location you specify with the Xcode DSTROOT setting. Use the **productbuild --root** option to specify that destination root directory and its install path. You can specify additional requirements using a PRE-INSTALL REQUIREMENTS PROPERTY LIST. When you specify a root, **productbuild** automatically creates a component package, much like pkgbuild(1), and synthesizes a distribution file.

4. Create a product archive using a distribution file. If you have a distribution file, use the **--distribution** option to specify the path to it, and the **--package-path** option to specify the directory where the component packages are found

:

NAME

productbuild — Build a product archive for the macOS Installer or the Mac App Store.

SYNOPSIS

productbuild [--product requirements-plist] {**--component** component-path [install-path]} product-output-path
productbuild {**--content** content-path} product-output-path
productbuild [--product requirements-plist] {**--root** root-path install-path} product-output-path
productbuild [options] **--distribution** dist-path [**--package-path** search-path] product-output-path
productbuild **--synthesize** [--product requirements-plist] {**--package** pkg-path} distribution-output-path

DESCRIPTION

NAME

productbuild — Build a product archive for the macOS Installer or the Mac App Store.

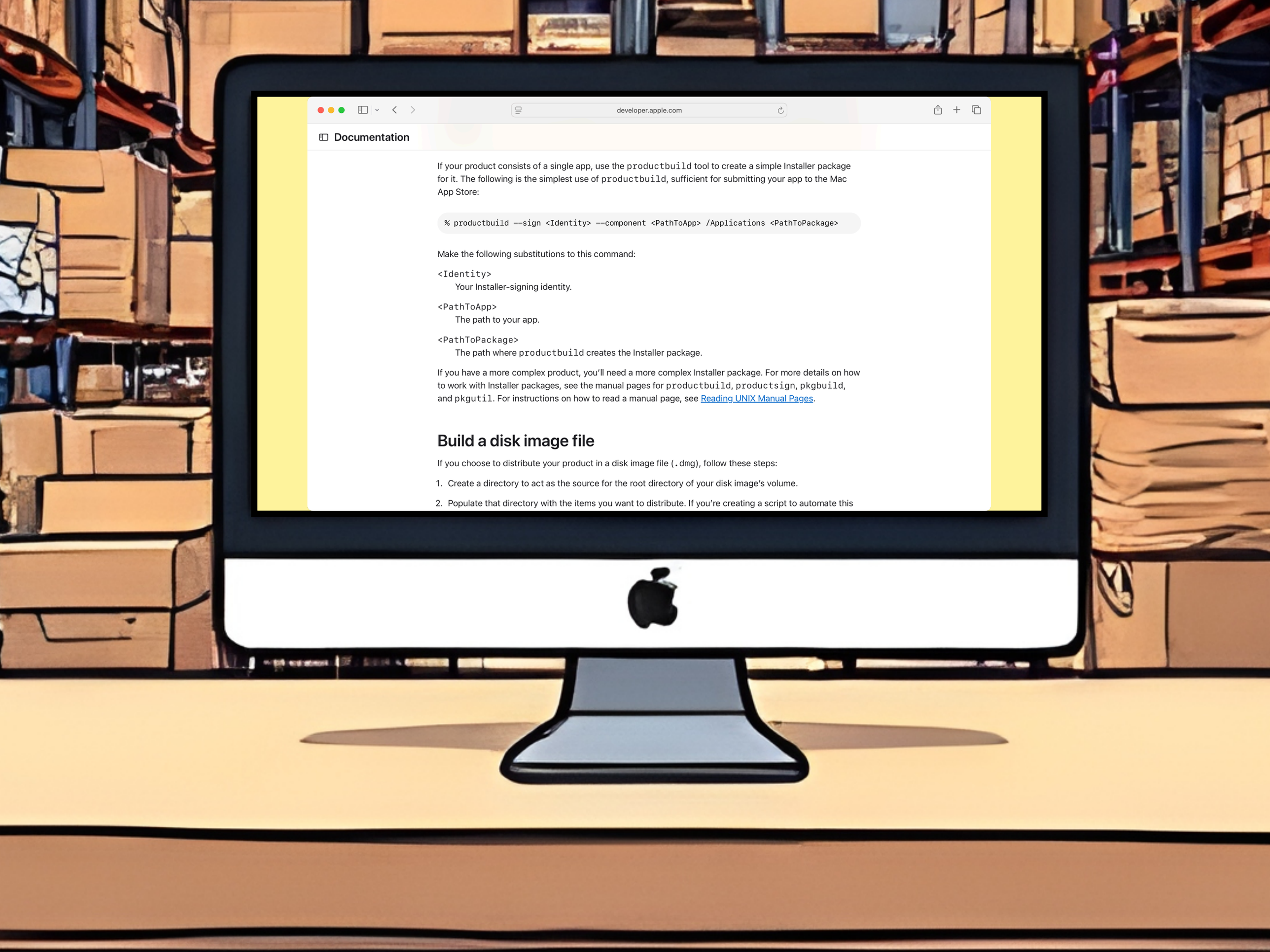
distribution file.

2. Create a product archive for in-app content. Specify in-app content using the **--content** option.

3. Create a product archive from a destination root. When you use xcodebuild(1) with the install action, the result is a destination root, either under /tmp, or in whatever location you specify with the Xcode DSTROOT setting. Use the **productbuild --root** option to specify that destination root directory and its install path. You can specify additional requirements using a PRE-INSTALL REQUIREMENTS PROPERTY LIST. When you specify a root, **productbuild** automatically creates a component package, much like pkgbuild(1), and synthesizes a distribution file.

4. Create a product archive using a distribution file. If you have a distribution file, use the **--distribution** option to specify the path to it, and the **--package-path** option to specify the directory where the component packages are found

:



developer.apple.com

Documentation

If your product consists of a single app, use the `productbuild` tool to create a simple Installer package for it. The following is the simplest use of `productbuild`, sufficient for submitting your app to the Mac App Store:

```
% productbuild --sign <Identity> --component <PathToApp> /Applications <PathToPackage>
```

Make the following substitutions to this command:

`<Identity>`
: Your Installer-signing identity.

`<PathToApp>`
: The path to your app.

`<PathToPackage>`
: The path where `productbuild` creates the Installer package.

If you have a more complex product, you'll need a more complex Installer package. For more details on how to work with Installer packages, see the manual pages for `productbuild`, `productsign`, `pkgbuild`, and `pkgutil`. For instructions on how to read a manual page, see Reading UNIX Manual Pages.

Build a disk image file

If you choose to distribute your product in a disk image file (`.dmg`), follow these steps:

1. Create a directory to act as the source for the root directory of your disk image's volume.
2. Populate that directory with the items you want to distribute. If you're creating a script to automate this

developer.apple.com

Documentation

If your product consists of a single app, use the `productbuild` tool to create a simple Installer package for it. The following is the simplest use of `productbuild`, sufficient for submitting your app to the Mac App Store:

```
% productbuild --sign <Identity> --component <PathToApp> /Applications <PathToPackage>
```

Make the following substitutions to this command:


```
% productbuild --sign <Identity> --component <PathToApp> /Applications <PathToPackage>
```

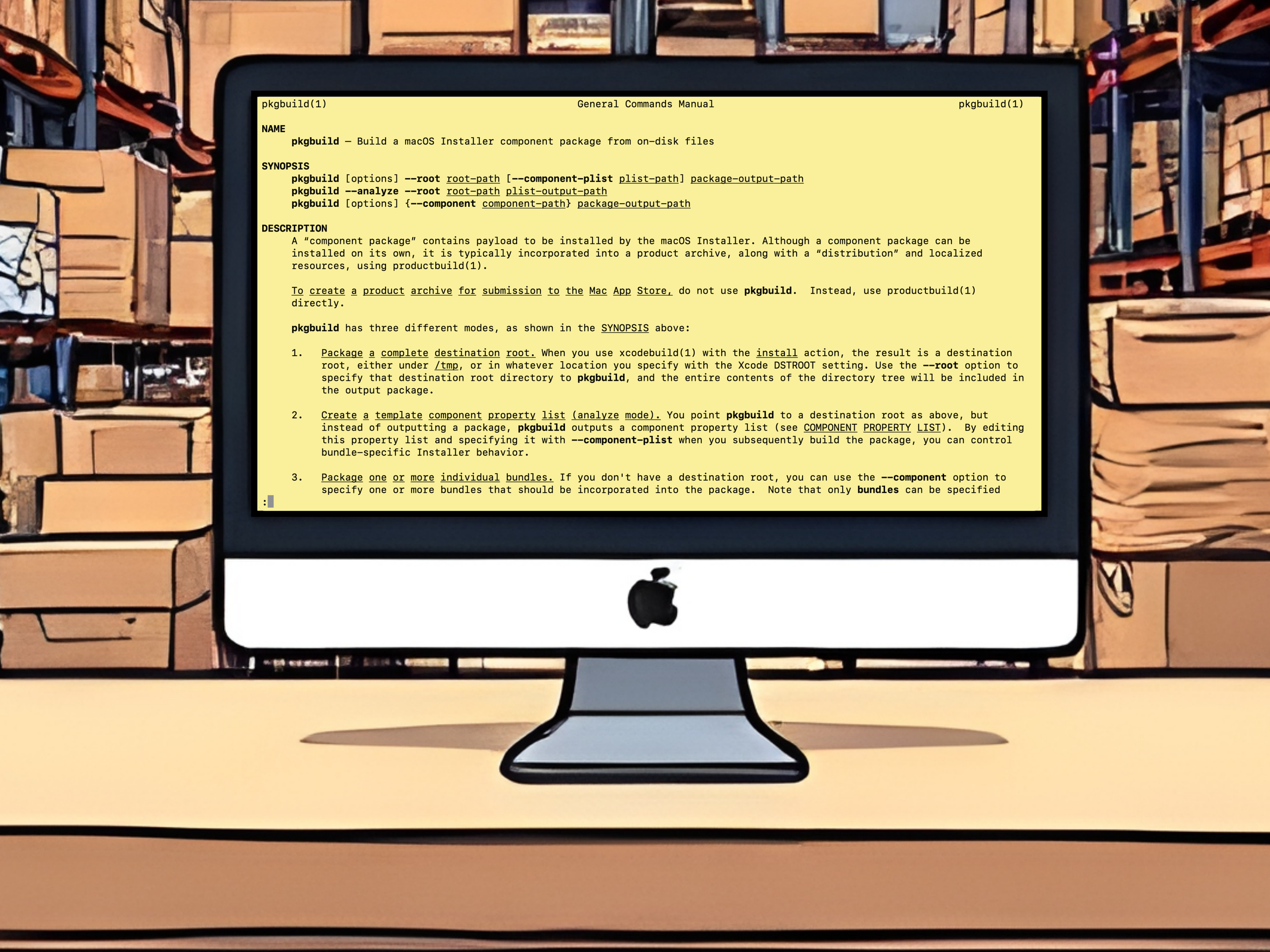

The path where `productbuild` creates the Installer package.

If you have a more complex product, you'll need a more complex Installer package. For more details on how to work with Installer packages, see the manual pages for `productbuild`, `productsign`, `pkgbuild`, and `pkgutil`. For instructions on how to read a manual page, see Reading UNIX Manual Pages.

Build a disk image file

If you choose to distribute your product in a disk image file (`.dmg`), follow these steps:

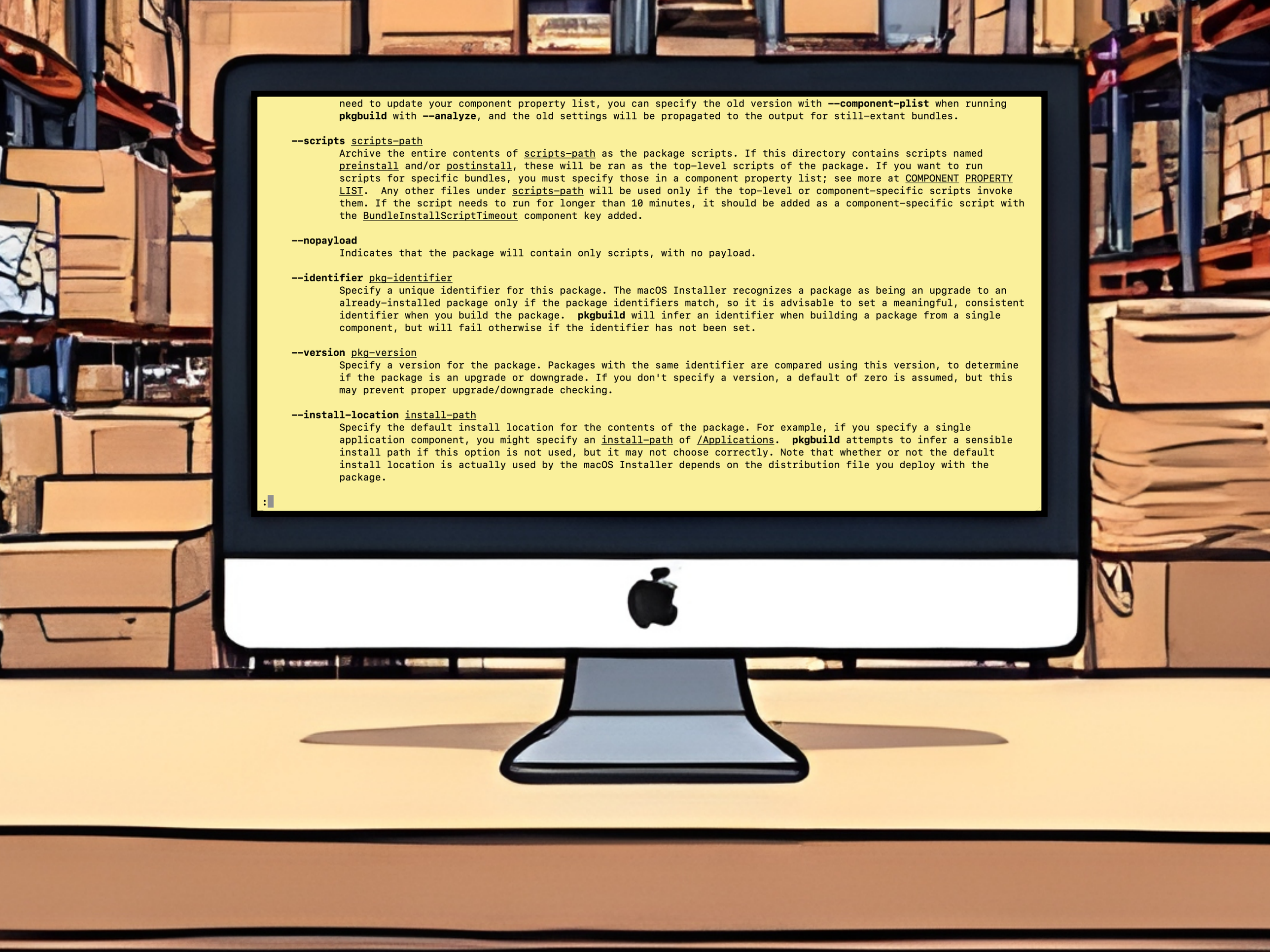
1. Create a directory to act as the source for the root directory of your disk image's volume.
2. Populate that directory with the items you want to distribute. If you're creating a script to automate this



```
pkgbuild(1)                          General Commands Manual                          pkgbuild(1)

NAME
     pkgbuild — Build a macOS Installer component package from on-disk files

SYNOPSIS
     pkgbuild [options] --root root-path [--component-plist plist-path] package-output-path
     pkgbuild --analyze --root root-path plist-output-path
     pkgbuild [options] {--component component-path} package-output-path

DESCRIPTION
     A "component package" contains payload to be installed by the macOS Installer. Although a component package can be
     installed on its own, it is typically incorporated into a product archive, along with a "distribution" and localized
     resources, using productbuild(1).

     To create a product archive for submission to the Mac App Store, do not use pkgbuild.  Instead, use productbuild(1)
     directly.

     pkgbuild has three different modes, as shown in the SYNOPSIS above:

     1.   Package a complete destination root. When you use xcodebuild(1) with the install action, the result is a destination
          root, either under /tmp, or in whatever location you specify with the Xcode DSTROOT setting. Use the --root option to
          specify that destination root directory to pkgbuild, and the entire contents of the directory tree will be included in
          the output package.

     2.   Create a template component property list (analyze mode). You point pkgbuild to a destination root as above, but
          instead of outputting a package, pkgbuild outputs a component property list (see COMPONENT PROPERTY LIST).  By editing
          this property list and specifying it with --component-plist when you subsequently build the package, you can control
          bundle-specific Installer behavior.

     3.   Package one or more individual bundles. If you don't have a destination root, you can use the --component option to
          specify one or more bundles that should be incorporated into the package.  Note that only bundles can be specified
:
```

pkgbuild(1) General Commands Manual pkgbuild(1)
NAME
pkgbuild — Build a macOS Installer component package from on-disk files
SYNOPSIS
pkgbuild [options] --root root-path [--component-plist plist-path] package-output-path
pkgbuild --analyze --root root-path plist-output-path
pkgbuild [options] {--component component-path} package-output-path
DESCRIPTION
A "component package" contains payload to be installed by the macOS Installer. Although a component package can be installed on its own, it is typically incorporated into a product archive, along with a "distribution" and localized resources, using productbuild(1).
pkgbuild — Build a macOS Installer component package from on-disk files
1. Package a complete destination root. When you use xcodebuild(1) with the install action, the result is a destination root, either under /tmp, or in whatever location you specify with the Xcode DSTROOT setting. Use the --root option to specify that destination root directory to pkgbuild, and the entire contents of the directory tree will be included in the output package.
2. Create a template component property list (analyze mode). You point pkgbuild to a destination root as above, but instead of outputting a package, pkgbuild outputs a component property list (see COMPONENT PROPERTY LIST). By editing this property list and specifying it with --component-plist when you subsequently build the package, you can control bundle-specific Installer behavior.
3. Package one or more individual bundles. If you don't have a destination root, you can use the --component option to specify one or more bundles that should be incorporated into the package. Note that only bundles can be specified
:

pkgbuild(1) General Commands Manual pkgbuild(1)
NAME
pkgbuild — Build a macOS Installer component package from on-disk files
SYNOPSIS
pkgbuild [options] --root root-path [--component-plist plist-path] package-output-path
pkgbuild --analyze --root root-path plist-output-path
pkgbuild [options] {--component component-path} package-output-path
DESCRIPTION
A "component package" contains payload to be installed by the macOS Installer. Although a component package can be installed on its own, it is typically incorporated into a product archive, along with a "distribution" and localized resources, using productbuild(1).
pkgbuild — Build a macOS Installer component package from on-disk files
pkgbuild has three different modes, as shown in the SYNOPSIS above.
1. Package a complete destination root. When you use xcodebuild(1) with the install action, the result is a destination root, either under /tmp, or in whatever location you specify with the Xcode DSTROOT setting. Use the --root option to
A "component package" contains payload to be installed by the macOS Installer. Although a component package can be installed on its own, it is typically incorporated into a product archive, along with a "distribution" and localized resources, using productbuild(1).
bundle-specific Installer behavior.
3. Package one or more individual bundles. If you don't have a destination root, you can use the --component option to specify one or more bundles that should be incorporated into the package. Note that only bundles can be specified
:



need to update your component property list, you can specify the old version with **--component-plist** when running **pkgbuild** with **--analyze**, and the old settings will be propagated to the output for still-extant bundles.

--scripts scripts-path
Archive the entire contents of scripts-path as the package scripts. If this directory contains scripts named preinstall and/or postinstall, these will be ran as the top-level scripts of the package. If you want to run scripts for specific bundles, you must specify those in a component property list; see more at COMPONENT PROPERTY LIST. Any other files under scripts-path will be used only if the top-level or component-specific scripts invoke them. If the script needs to run for longer than 10 minutes, it should be added as a component-specific script with the BundleInstallScriptTimeout component key added.

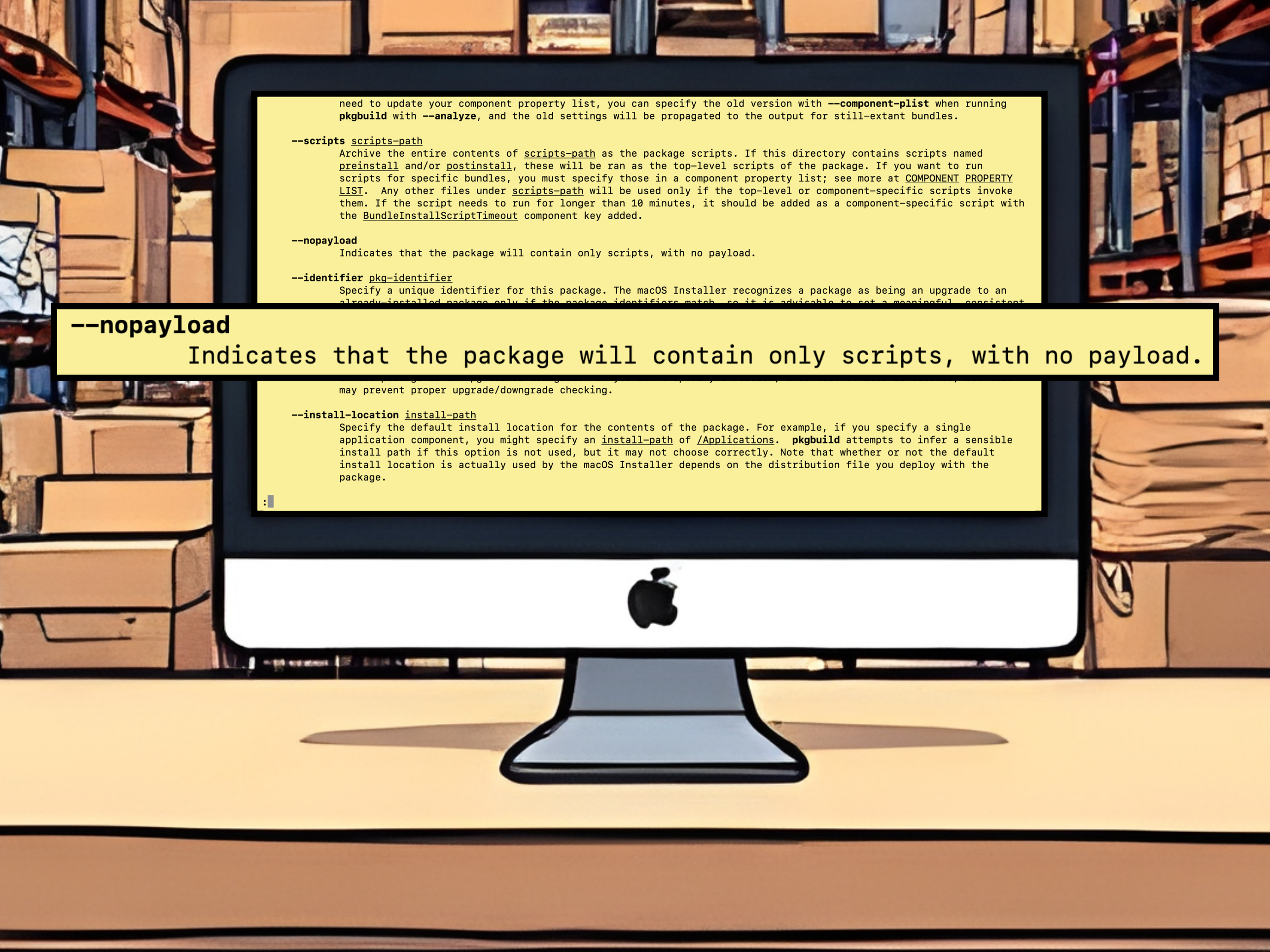
--nopayload
Indicates that the package will contain only scripts, with no payload.

--identifier pkg-identifier
Specify a unique identifier for this package. The macOS Installer recognizes a package as being an upgrade to an already-installed package only if the package identifiers match, so it is advisable to set a meaningful, consistent identifier when you build the package. **pkgbuild** will infer an identifier when building a package from a single component, but will fail otherwise if the identifier has not been set.

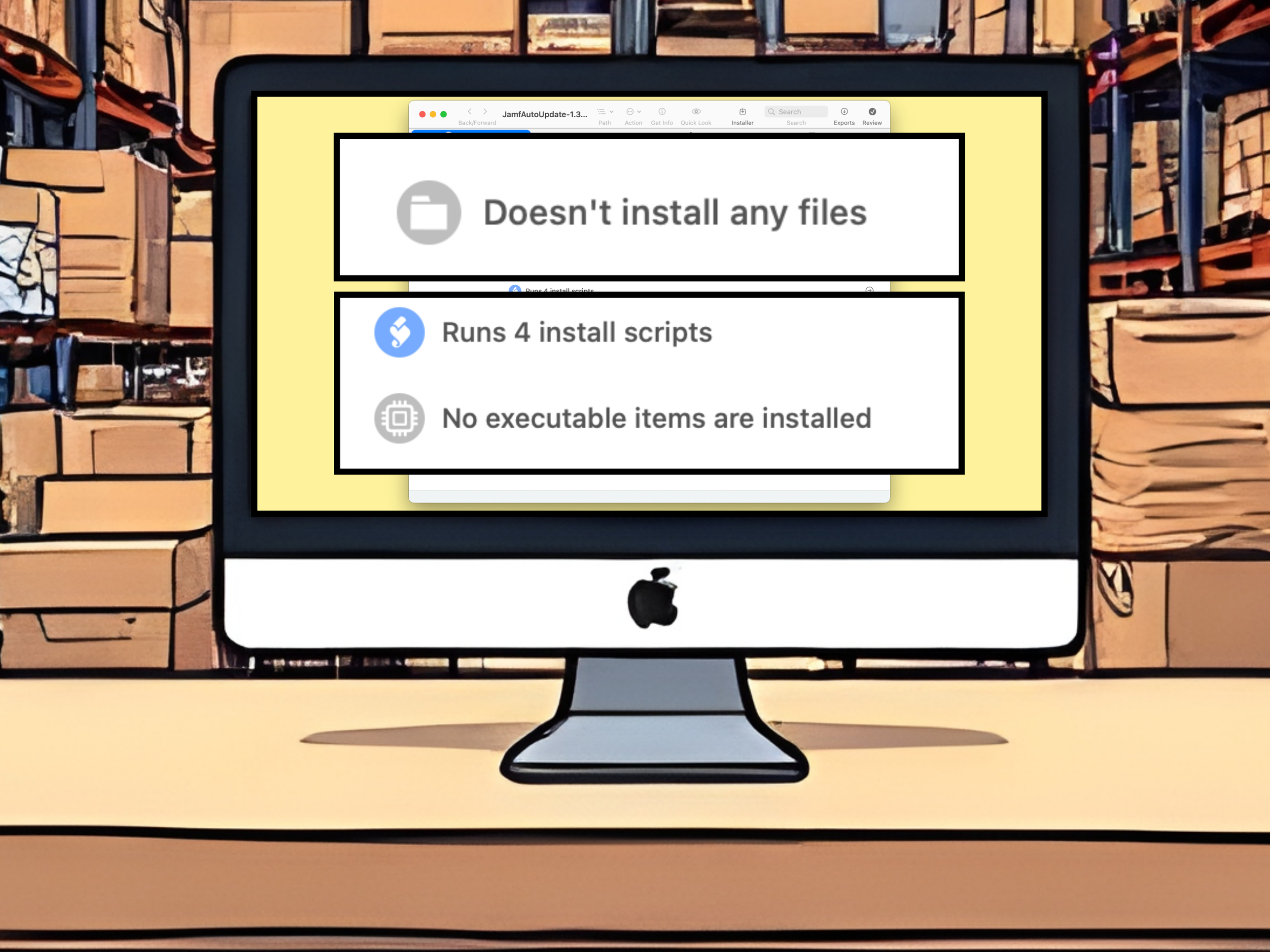
--version pkg-version
Specify a version for the package. Packages with the same identifier are compared using this version, to determine if the package is an upgrade or downgrade. If you don't specify a version, a default of zero is assumed, but this may prevent proper upgrade/downgrade checking.

--install-location install-path
Specify the default install location for the contents of the package. For example, if you specify a single application component, you might specify an install-path of /Applications. **pkgbuild** attempts to infer a sensible install path if this option is not used, but it may not choose correctly. Note that whether or not the default install location is actually used by the macOS Installer depends on the distribution file you deploy with the package.

:

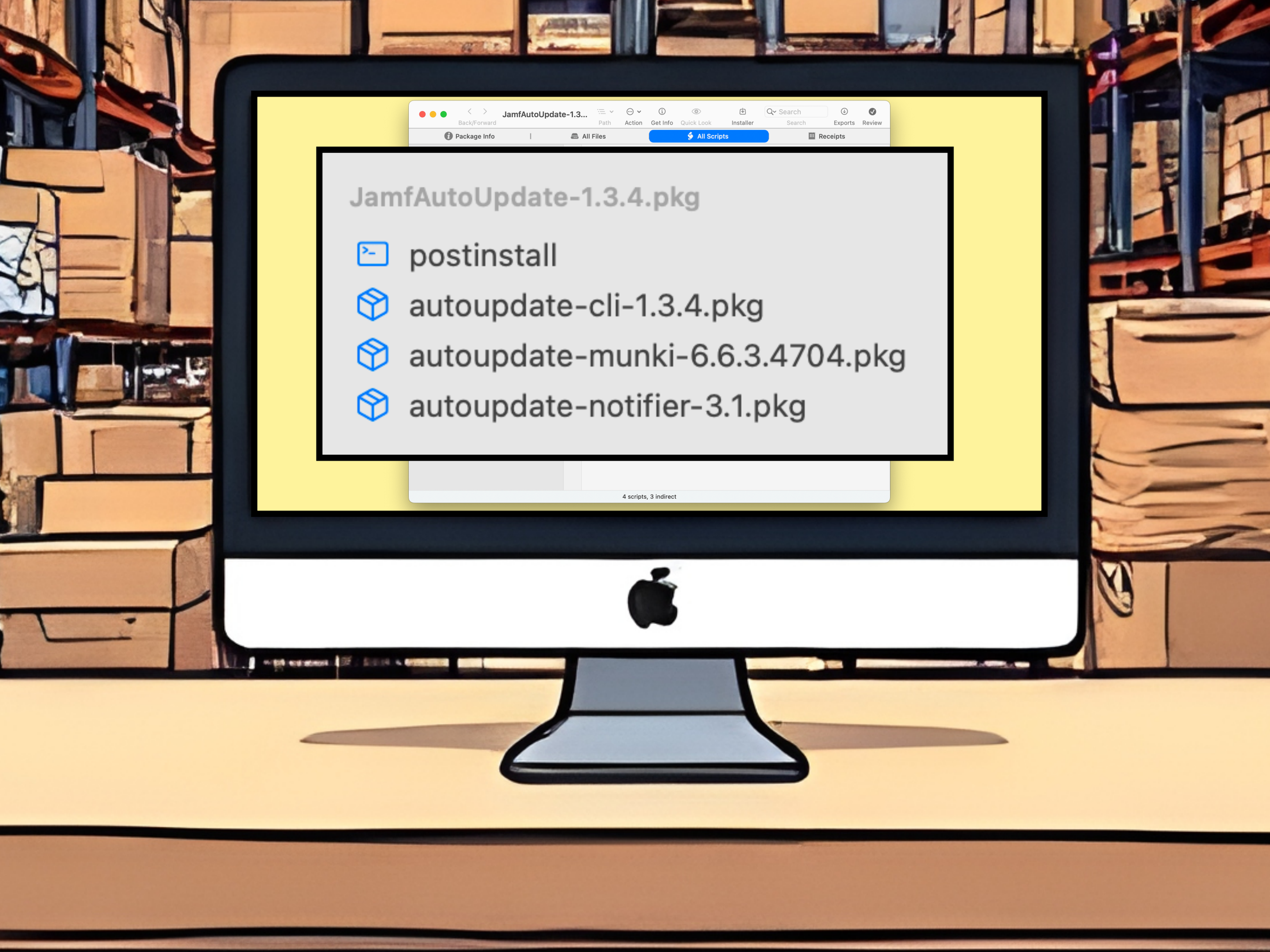

need to update your component property list, you can specify the old version with --component-plist when running pkgbuild with --analyze, and the old settings will be propagated to the output for still-extant bundles.
--scripts scripts-path
Archive the entire contents of scripts-path as the package scripts. If this directory contains scripts named preinstall and/or postinstall, these will be ran as the top-level scripts of the package. If you want to run scripts for specific bundles, you must specify those in a component property list; see more at COMPONENT PROPERTY LIST. Any other files under scripts-path will be used only if the top-level or component-specific scripts invoke them. If the script needs to run for longer than 10 minutes, it should be added as a component-specific script with the BundleInstallScriptTimeout component key added.
--nopayload
Indicates that the package will contain only scripts, with no payload.
--identifier pkg-identifier
Specify a unique identifier for this package. The macOS Installer recognizes a package as being an upgrade to an
--nopayload
Indicates that the package will contain only scripts, with no payload.
may prevent proper upgrade/downgrade checking.
--install-location install-path
Specify the default install location for the contents of the package. For example, if you specify a single application component, you might specify an install-path of /Applications. pkgbuild attempts to infer a sensible install path if this option is not used, but it may not choose correctly. Note that whether or not the default install location is actually used by the macOS Installer depends on the distribution file you deploy with the package.
:

JamfAutoUpdate-1.3...
Back/Forward
Path
Action
Get Info
Quick Look
Installer
Search
Search
Exports
Review
Package Info
All Files
All Scripts
Receipts
JamfAutoUpdate-1.3.4.pkg
DEVELOPER ID SOFTWARE • COMPONENT PACKAGE
Doesn't install any files
DEVELOPER ID
Signed by "Developer ID Installer: JAMF Software (483DWKW443)"
Notarized by Apple — Stapled Ticket
Runs 4 install scripts
No executable items are installed
No executable items request entitlements
Downloaded by Safari — 28 January 2025 at 16:01
Didn't find any issues for review


JamfAutoUpdate-1.3...
Back/Forward
Path
Action
Get Info
Quick Look
Installer
Search
Search
Exports
Review
Doesn't install any files
Runs 4 install scripts
No executable items are installed

JamfAutoUpdate-1.3...
Back/Forward
Path
Action
Get Info
Quick Look
Installer
Search
Exports
Review
Package Info
All Files
All Scripts
Receipts
Name
Date Modified
Size
Kind
Name --
Kind --
Executable --
Size --
Modified --
Owner --
Group --
Permissions
User
Other
Type
Group
Version --
Identifier --
Requires --
Entitlements
No Entitlements
All Files
No items

JamfAutoUpdate-1.3...
Back/Forward
Path
Action
Get Info
Quick Look
Installer
Search
Search
Exports
Review
Package Info
All Files
All Scripts
Receipts
JamfAutoUpdate-1.3.4.pkg
postinstall
autoupdate-cli-1.3.4.pkg
autoupdate-munki-6.6.3.4704.pkg
autoupdate-notifier-3.1.pkg
No script selected.
4 scripts, 3 indirect


JamfAutoUpdate-1.3...
Back/Forward
Path
Action
Get Info
Quick Look
Installer
Search
Exports
Review
Package Info
All Files
All Scripts
Receipts
JamfAutoUpdate-1.3.4.pkg
postinstall
autoupdate-cli-1.3.4.pkg
autoupdate-munki-6.6.3.4704.pkg
autoupdate-notifier-3.1.pkg
4 scripts, 3 indirect

10

- `pkgbuild` builds a macOS Installer *"component package"*. A *component package* contains a payload to be installed by the macOS Installer. Throughout the man pages, component packages are also referred to as *"packages"*. Although a component package can be installed on its own, it is typically incorporated into a *"product archive"*, along with a distribution file and localized resources, using `productbuild`.
- `productbuild` builds a *"product archive"* for the macOS Installer or the Mac App Store. A *product archive* is also a file to be used by the macOS Installer, but it can contain multiple *component packages* to be installed. And by using `productbuild` to create a product archive, you can add elements to customize the installation process for users (such as your own text, license agreement, localized resources, logo etc.). You will see product archives also referred to as *"distribution packages"* [1] [2] and *"installer packages"* [3]. If you want to upload your bundle/app to the Mac App Store, then you need to create a product archive for it with `productbuild` rather than only using `pkgbuild`.

Confusingly the output files (*component packages* and *product archives*) of the two commands both have the same extension (`.pkg`). But you can find out more about a `.pkg` by looking at the contents, which you can see by right clicking it and selecting *Show Package Contents* or by running `pkgutil --expand my-package.pkg path/to/expand`. [4]

Hopefully some of those links provide you with some more reading material about what the commands do. This blogpost details more about using the two commands, and this is a good blog post outlining how the commands can be used to create a custom `.pkg` installer for a macOS application.

Share Improve this answer Follow

answered Feb 17, 2023 at 14:08
nating
347 ● 2 ● 7 ● 14

Add a comment

Der Flounder

Seldom updated, occasionally insightful.

Preparing installer packages for installation using MDM commands

October 24, 2023 rtrouton Go to comments Leave a comment

An issue that some Mac admins have had to deal with is that their system management tool is using MDM commands to install installer packages. This usually applies if the system management tool does not have an agent installed on the managed Macs and instead is using only MDM for management.

In those cases, installer packages must have the following attributes for a successful installation via MDM command:

1. Signed with an Apple Developer ID Installer certificate
2. Be a distribution installer package

For criteria #2, this references the fact that there are two kinds of modern installer packages for macOS:

- **Component packages**: these are the standard type of installer package, which contain an archive of files to install and the information on where the files should be installed.
- **Distribution packages**: These packages can contain one or more component packages, and may also include additional resources to customize and control the user interface shown in the Installer application.

Both component and distribution packages use the same icon by default, so you can't tell the difference by visually looking at an installer package. However, you can use the xar command line tool to check inside an installer package and list the files stored inside. The reason why this helps is that all distribution packages will have a file inside named **Distribution** and component packages will not.

To check an installer package to see if it is a distribution package, use the command shown below:

RSS feed

October 2023

M	T	W	T	F	S	S
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

« Sep Nov »

Recent Comments

Peter on rsync replaced with openrsync...

postpractically782db... on Using sysdiagnose logarchive f...

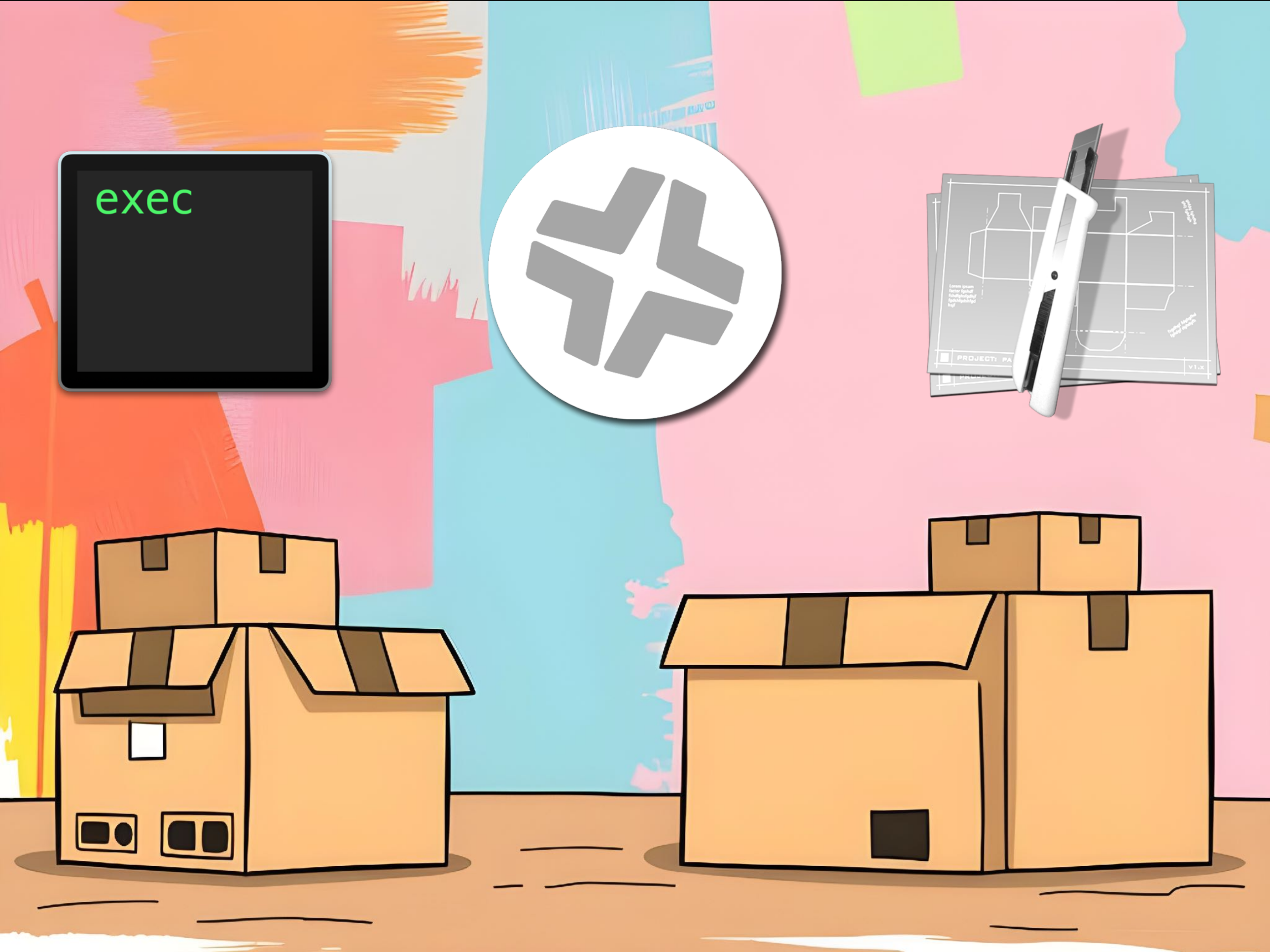
staze on Using sysdiagnose logarchive f...

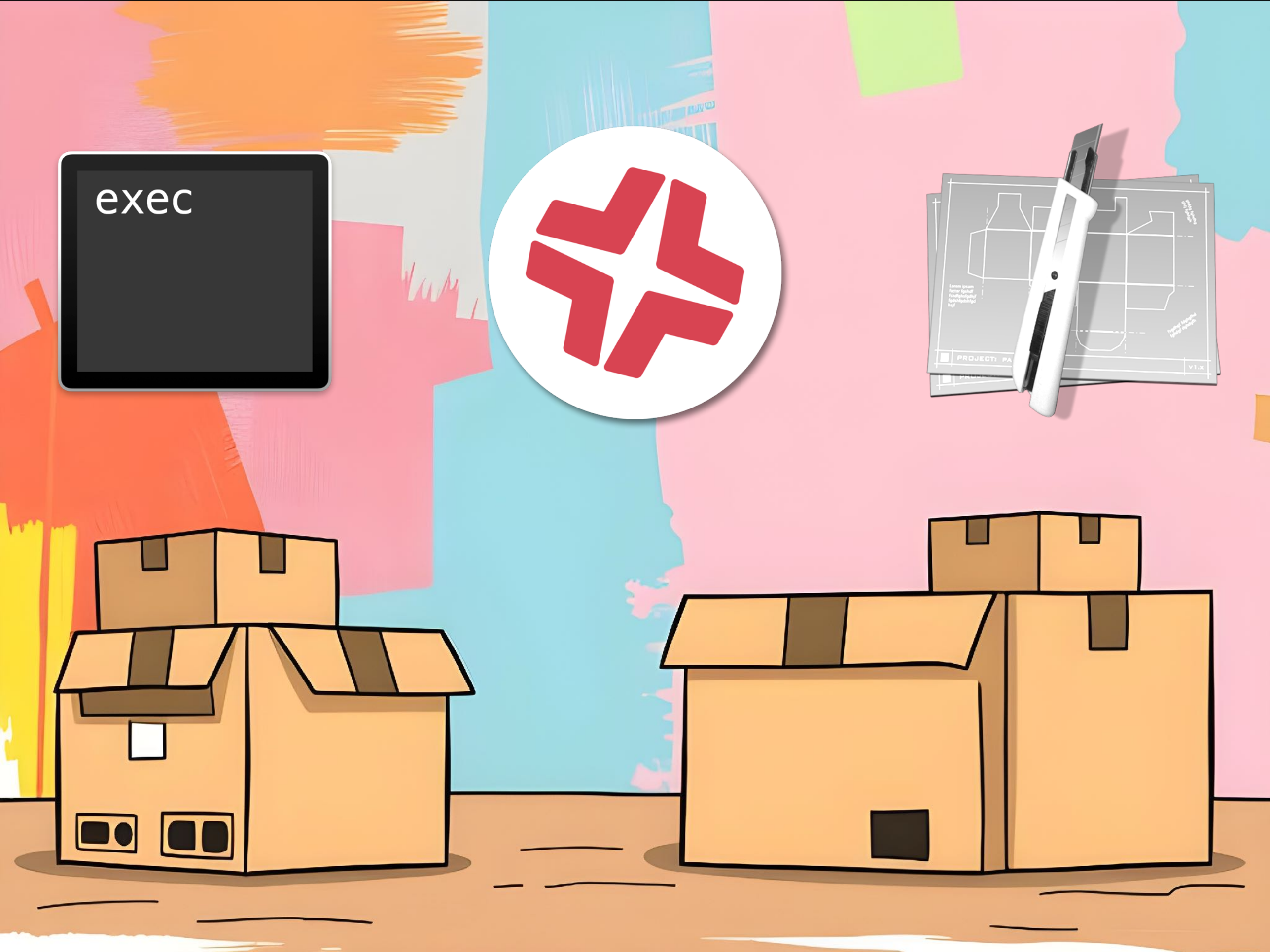
rtrouton on Using sysdiagnose logarchive f...

staze on Using sysdiagnose logarchive f...

Categories

Select Category


exec

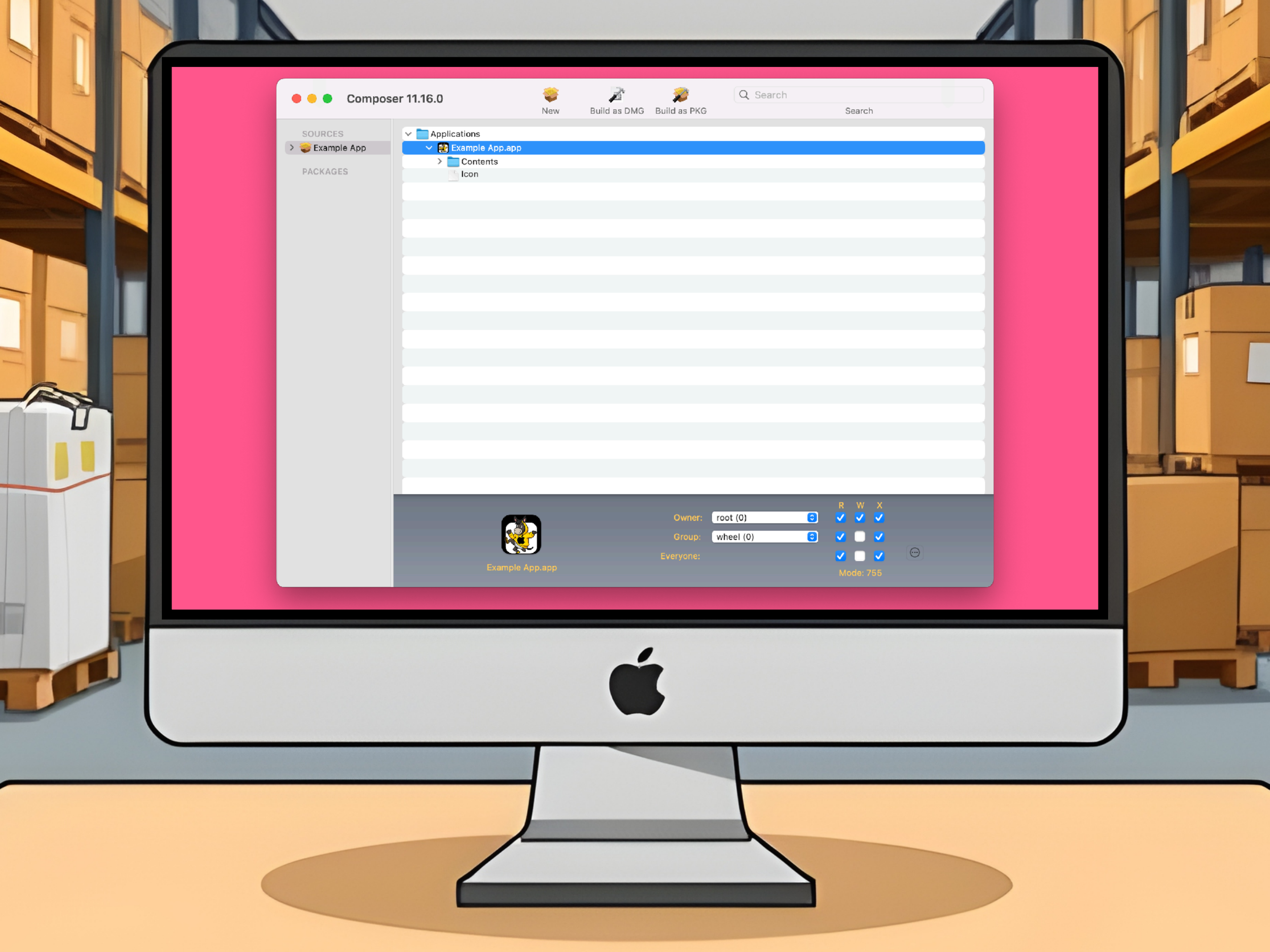

exec

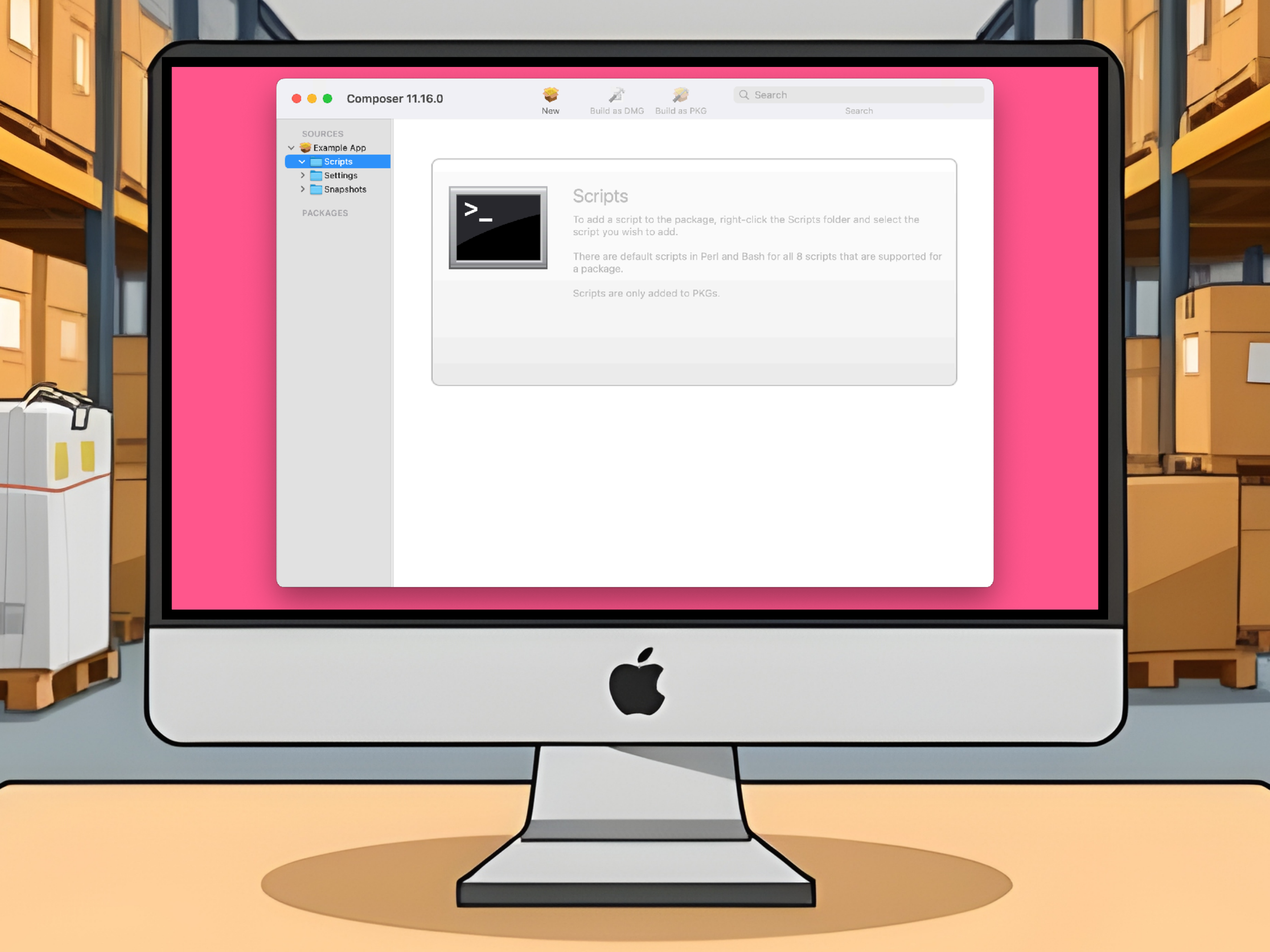
Friends don't let
friends create
packages from
Composer
created
snapshots.

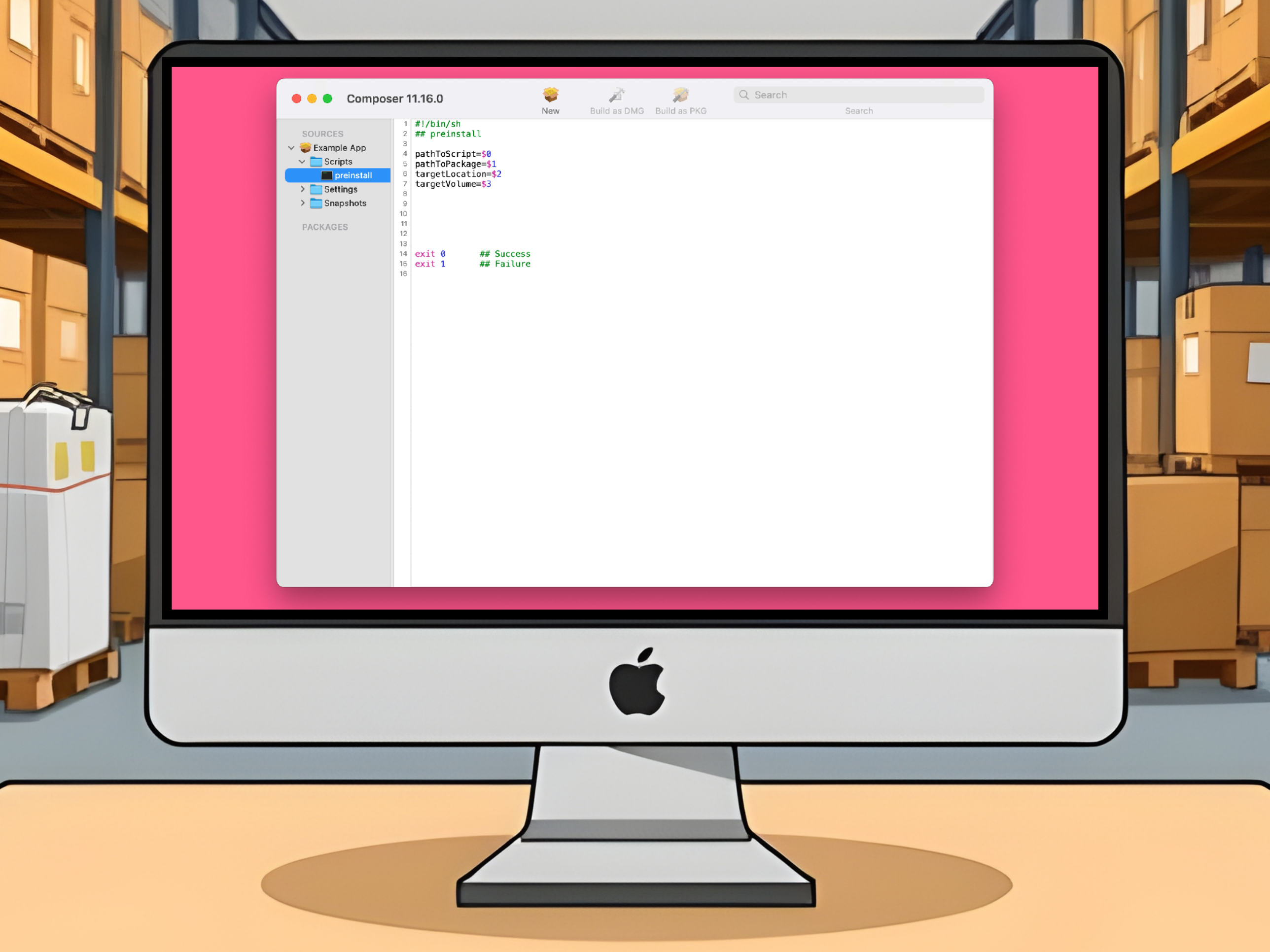
Friends don't let
friends create
packages from
Composer
created
snapshots.

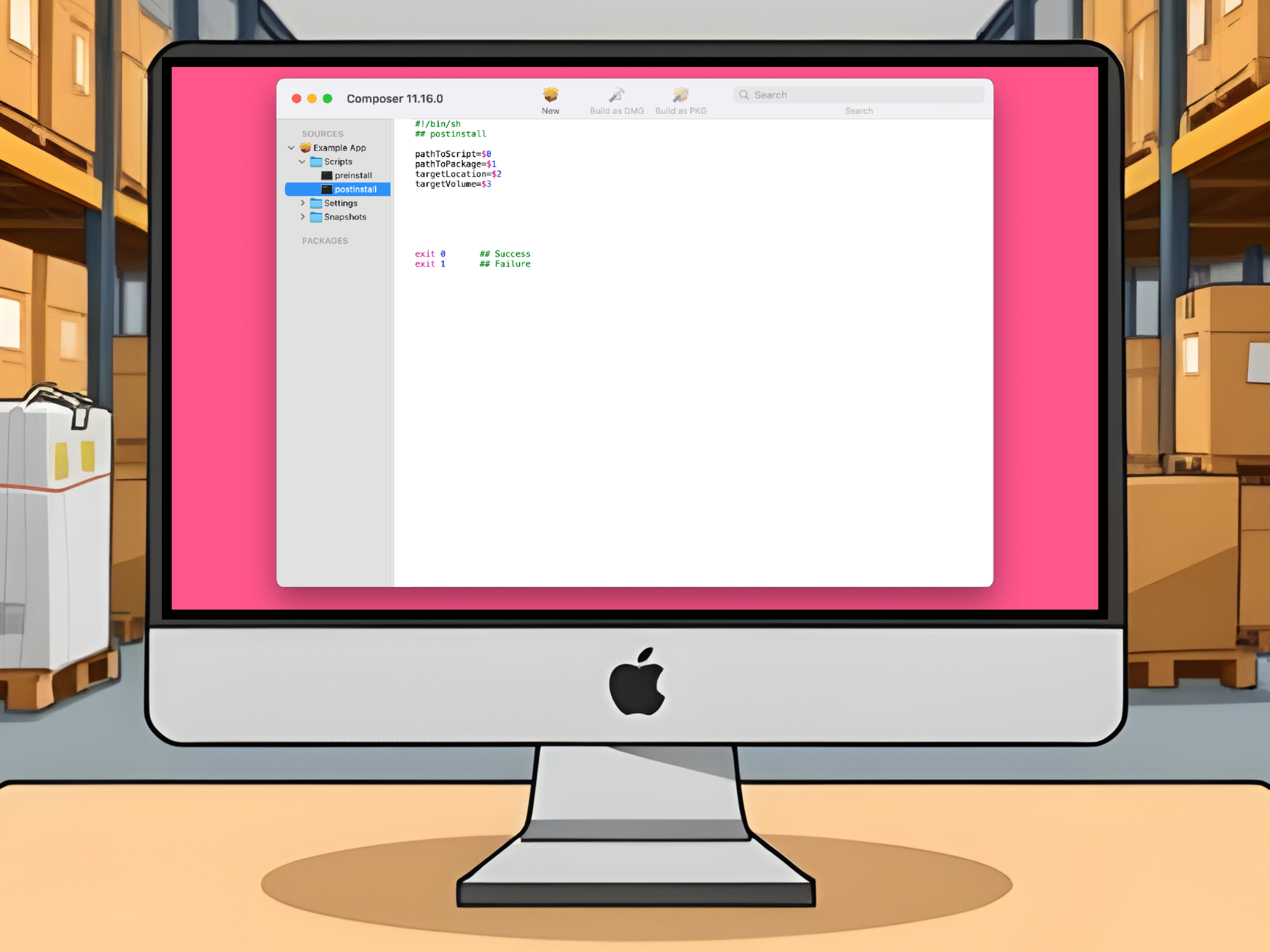
Composer 11.16.0
New
Build as DMG
Build as PKG
Search
Search
SOURCES
PACKAGES
Choose a method to create your package:
Monitoring Installation
Snapshot
Package Manifests
Pre-Installed Software
User Environment
New & Modified Snapshot
Monitor File System Changes
Normal Snapshot
Take a snapshot that watches for new files, as well as any files that are modified. A snapshot will be taken before and after the installation process, and then the two snapshots will be compared.
Cancel
Previous
Next

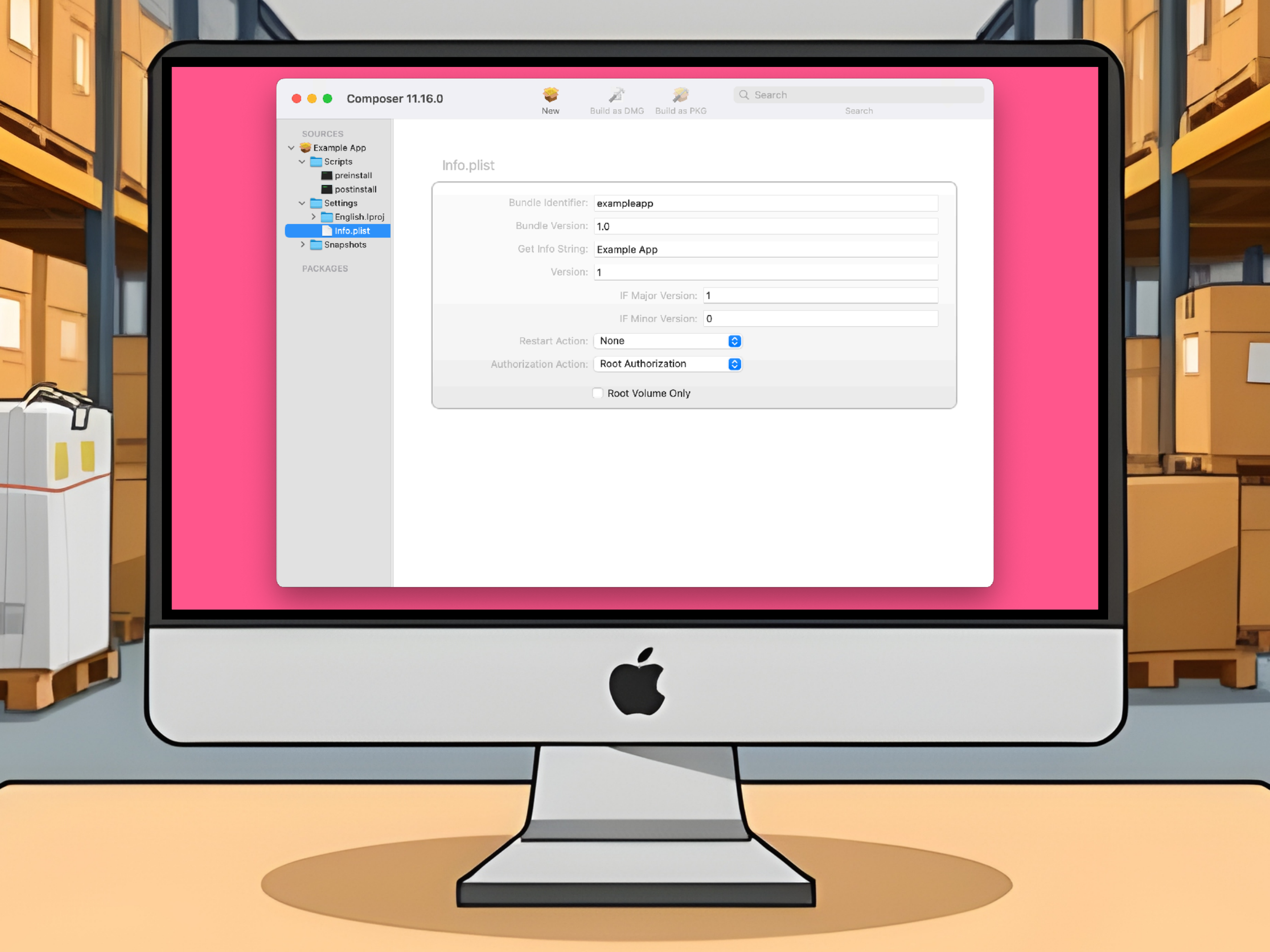
Composer 11.16.0
New
Build as DMG
Build as PKG
Search
Search
SOURCES
PACKAGES


Composer 11.16.0
New
Build as DMG
Build as PKG
Search
Search
SOURCES
Example App
PACKAGES
Applications
Example App.app
Contents
Icon
Example App.app
Owner:
root (0)
Group:
wheel (0)
Everyone:
R
W
X
Mode: 755


Composer 11.16.0
New
Build as DMG
Build as PKG
Search
Search
SOURCES
Example App
Scripts
Settings
Snapshots
PACKAGES
Scripts
To add a script to the package, right-click the Scripts folder and select the script you wish to add.
There are default scripts in Perl and Bash for all 8 scripts that are supported for a package.
Scripts are only added to PKGs.

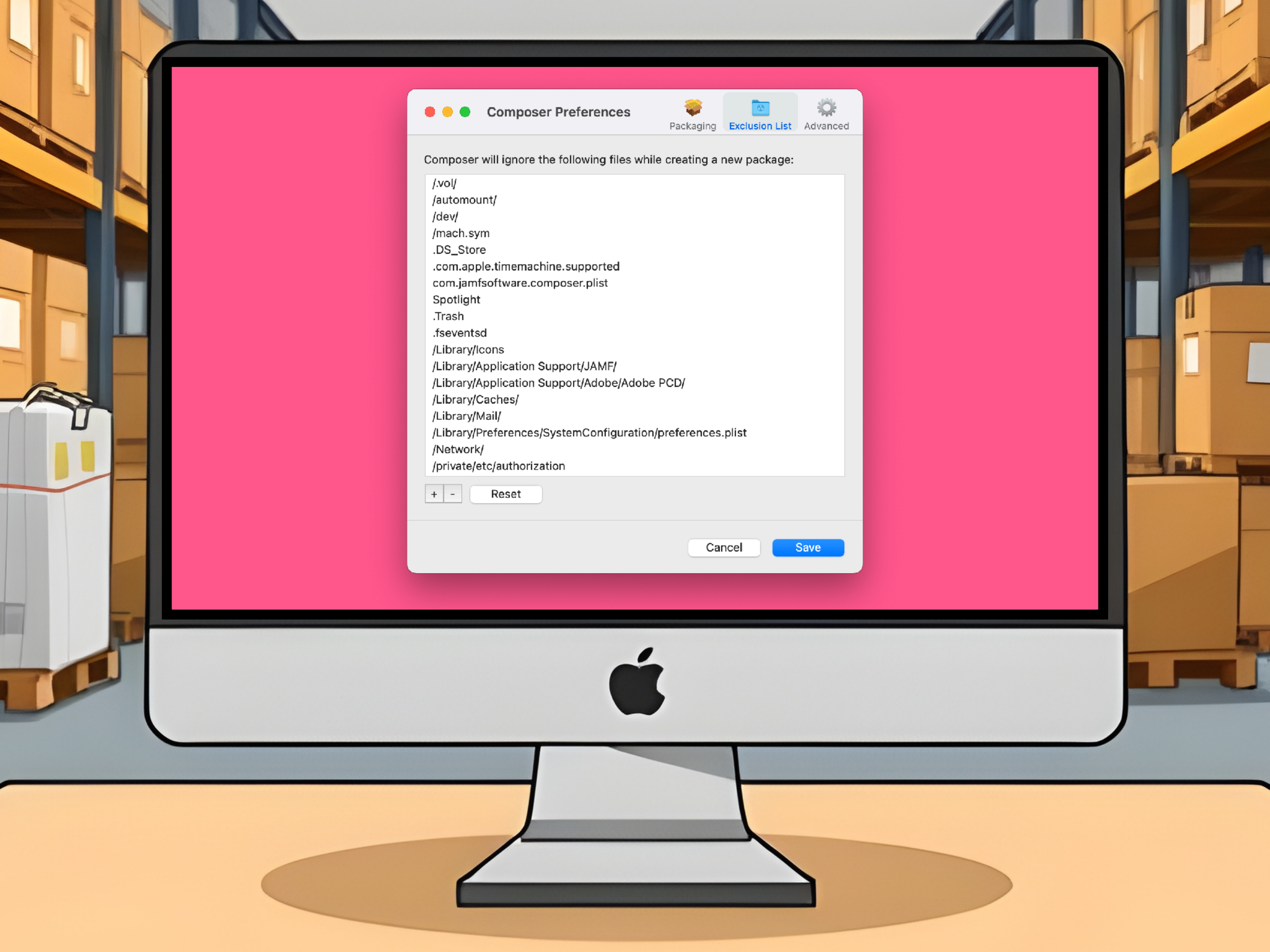

Composer 11.16.0
New
Build as DMG
Build as PKG
Search
Search
SOURCES
Example App
Scripts
preinstall
Settings
Snapshots
PACKAGES
#!/bin/sh
preinstall
pathToScript=$0
pathToPackage=$1
targetLocation=$2
targetVolume=$3
exit 0 ## Success
exit 1 ## Failure

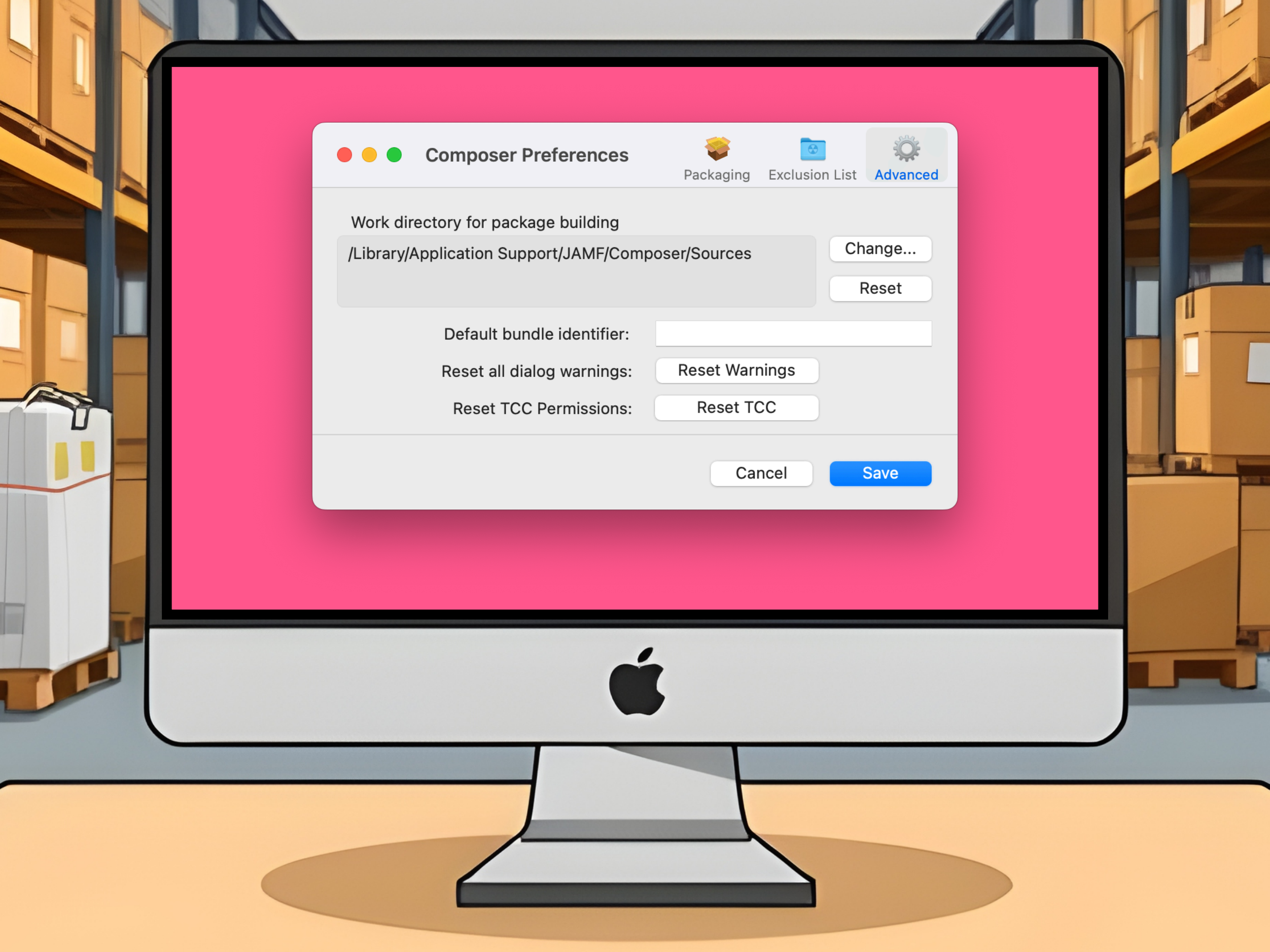

Composer 11.16.0
New
Build as DMG
Build as PKG
Search
Search
SOURCES
Example App
Scripts
preinstall
postinstall
Settings
Snapshots
PACKAGES
#!/bin/sh
postinstall
pathToScript=$0
pathToPackage=$1
targetLocation=$2
targetVolume=$3
exit 0 ## Success
exit 1 ## Failure

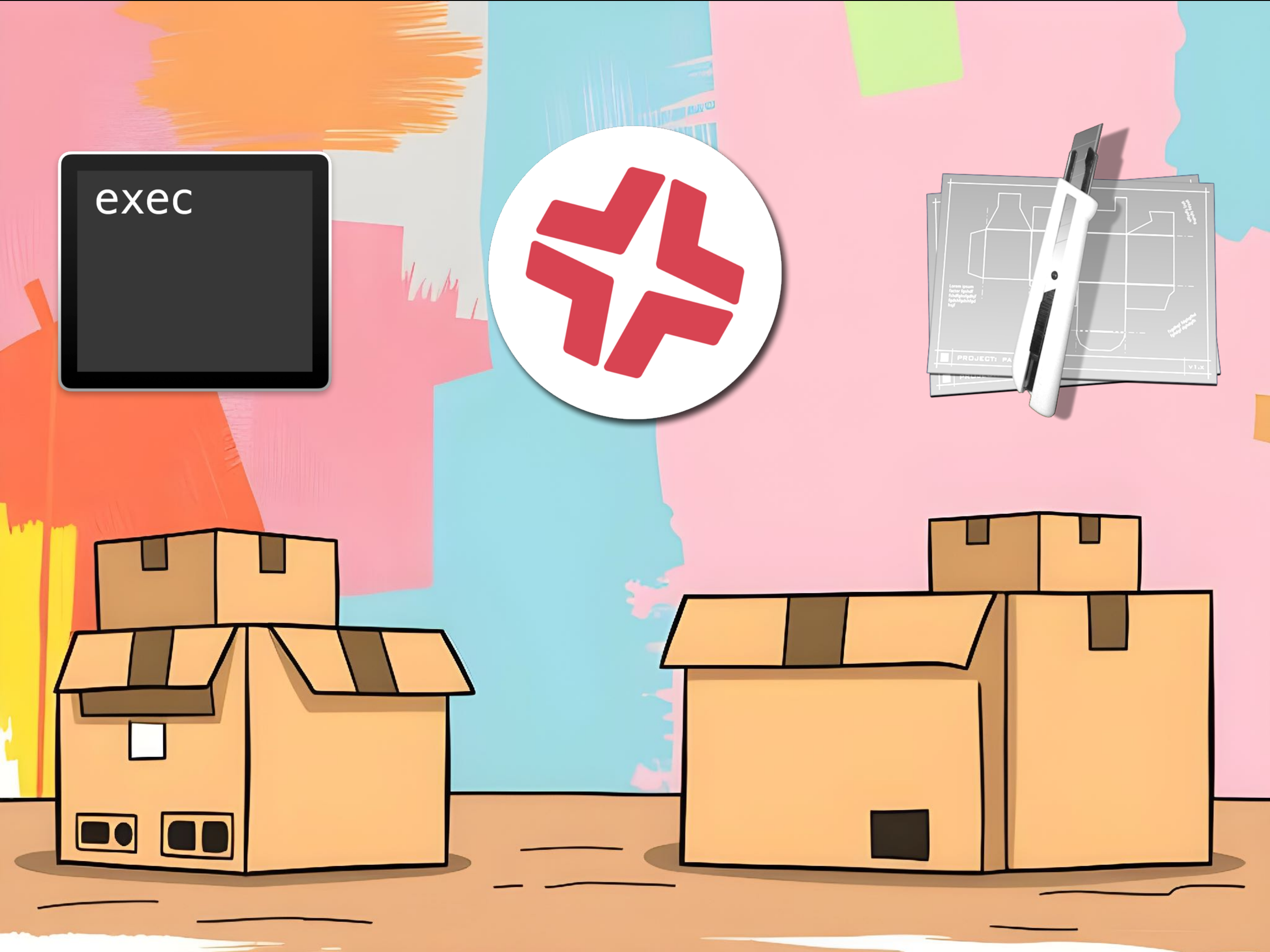

Composer 11.16.0
New
Build as DMG
Build as PKG
Search
Search
SOURCES
Example App
Scripts
preinstall
postinstall
Settings
English.lproj
Info.plist
Snapshots
PACKAGES
Info.plist
Bundle Identifier: exampleapp
Bundle Version: 1.0
Get Info String: Example App
Version: 1
IF Major Version: 1
IF Minor Version: 0
Restart Action: None
Authorization Action: Root Authorization
Root Volume Only

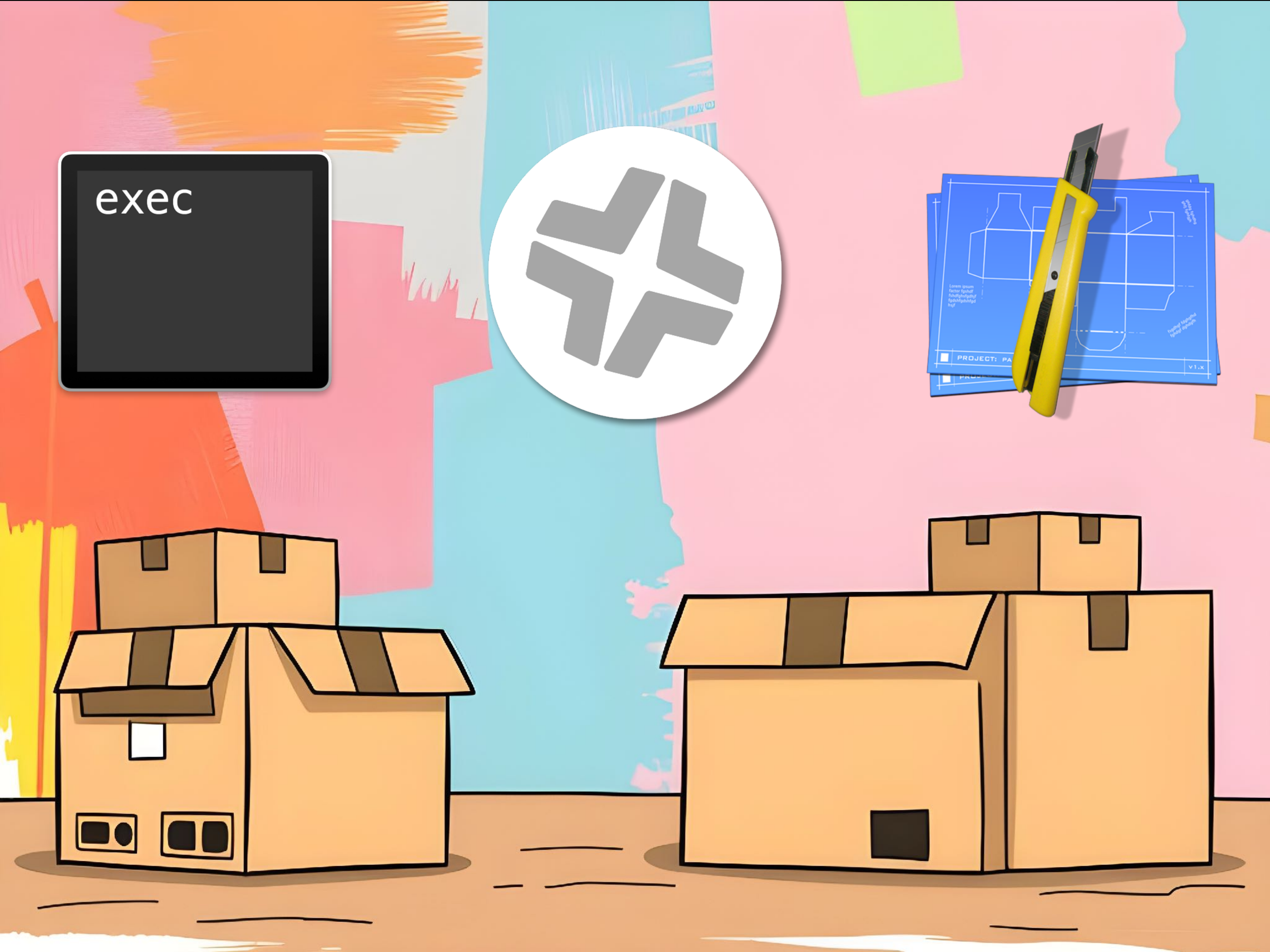
Composer 11.16.0
New
Build as DMG
Build as PKG
Search
Search
SOURCES
Example App
Scripts
preinstall
postinstall
Settings
English.lproj
Info.plist
Snapshots
PACKAGES
Applications
R W X
Owner:
Group:
Everyone:
Nothing Selected

Composer Preferences
Packaging
Exclusion List
Advanced
Build flat PKGs
Sign with:
Remove .DS_Store files in common locations
Scan images when building DMGs
Play sounds
Reveal in Finder when done
Executable Types in PKGs:
Automatically detect executable types
DMG Target Filesystem:
Prompt
Cancel
Save

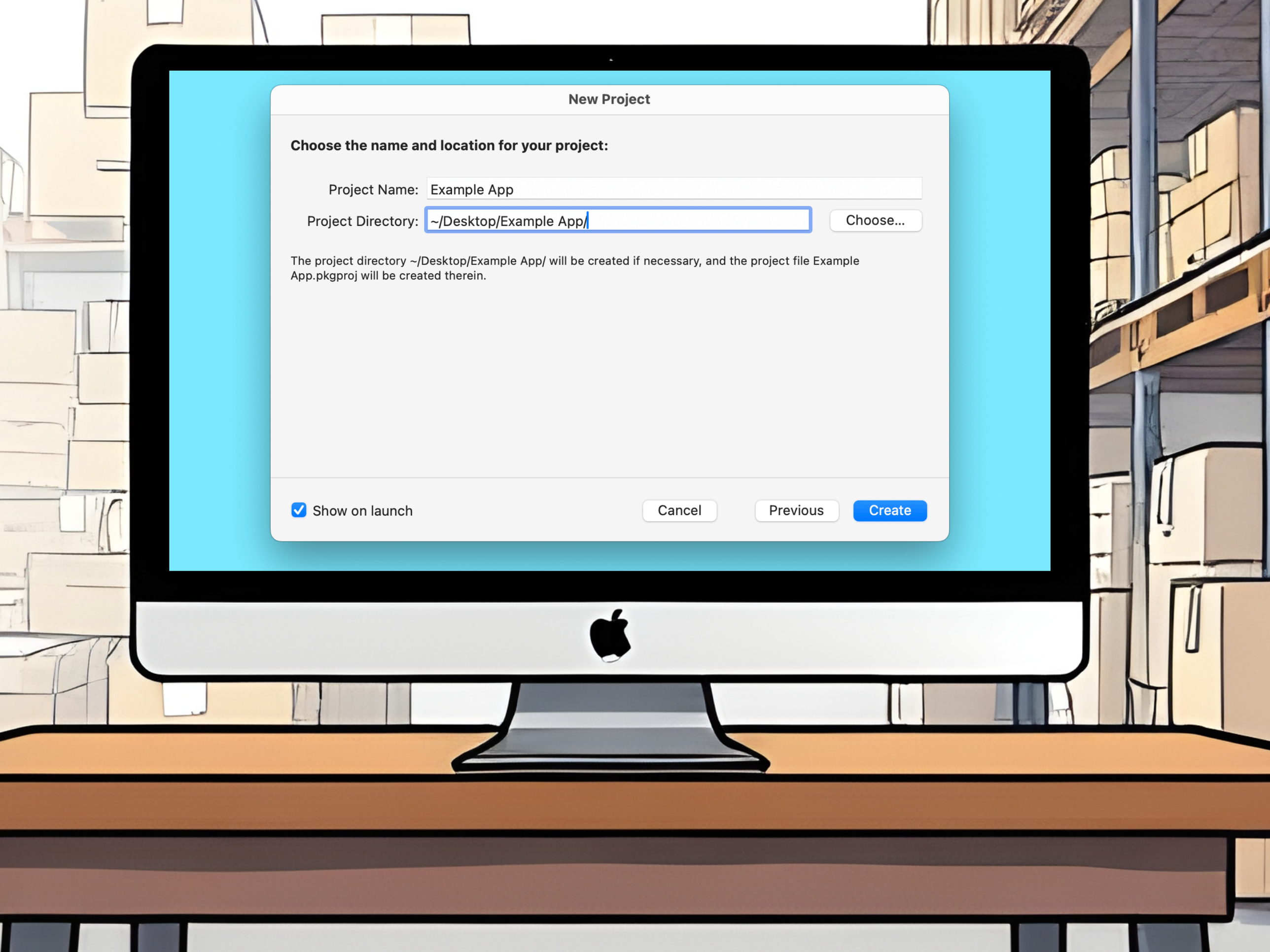

Composer Preferences
Packaging
Exclusion List
Advanced
Composer will ignore the following files while creating a new package:
/.vol/
/automount/
/dev/
/mach.sym
.DS_Store
.com.apple.timemachine.supported
com.jamfsoftware.composer.plist
Spotlight
.Trash
.fseventsd
/Library/Icons
/Library/Application Support/JAMF/
/Library/Application Support/Adobe/Adobe PCD/
/Library/Caches/
/Library/Mail/
/Library/Preferences/SystemConfiguration/preferences.plist
/Network/
/private/etc/authorization
+
-
Reset
Cancel
Save

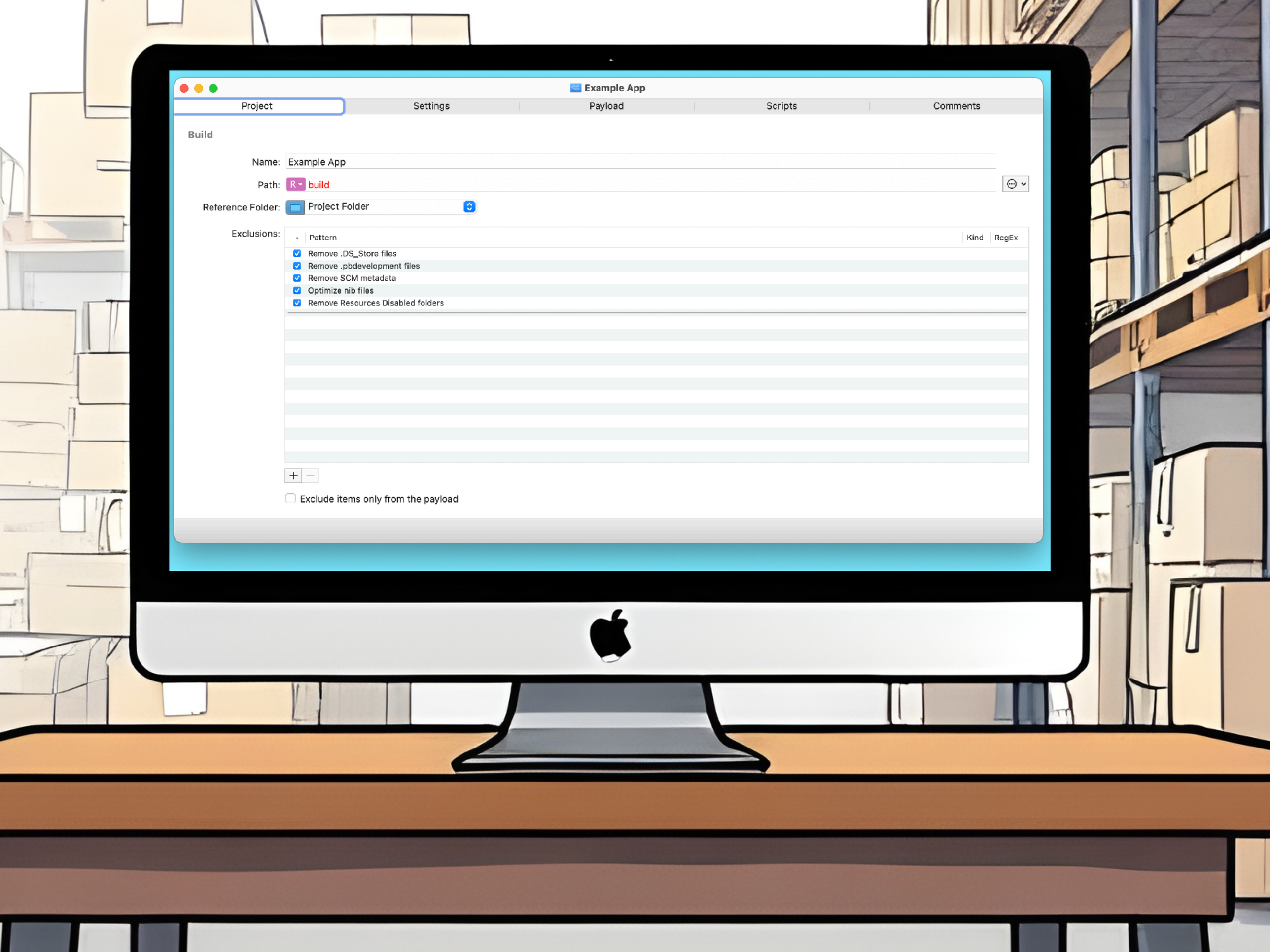

Composer Preferences
Packaging
Exclusion List
Advanced
Work directory for package building
/Library/Application Support/JAMF/Composer/Sources
Change...
Reset
Default bundle identifier:
Reset all dialog warnings:
Reset Warnings
Reset TCC Permissions:
Reset TCC
Cancel
Save

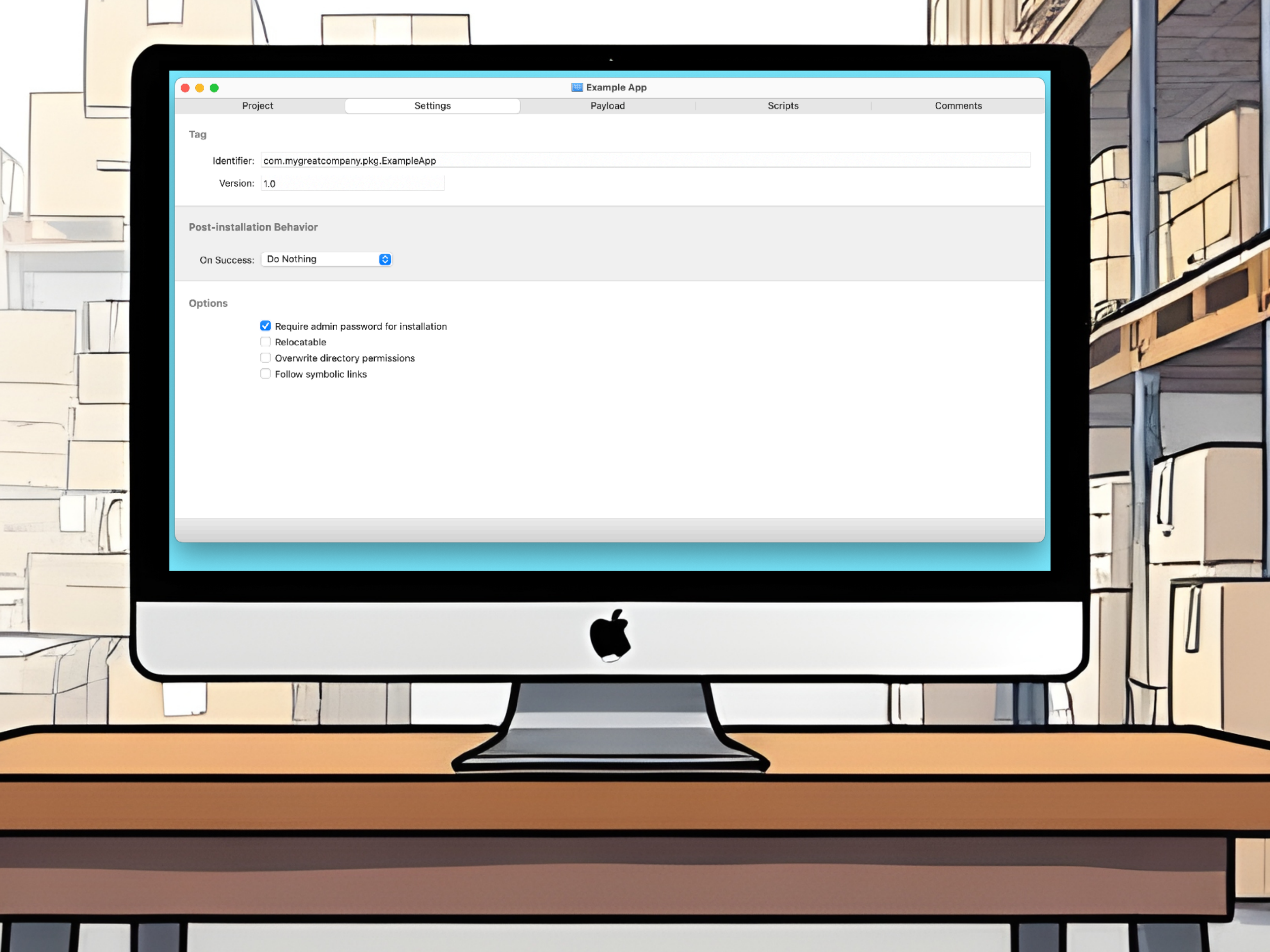

exec

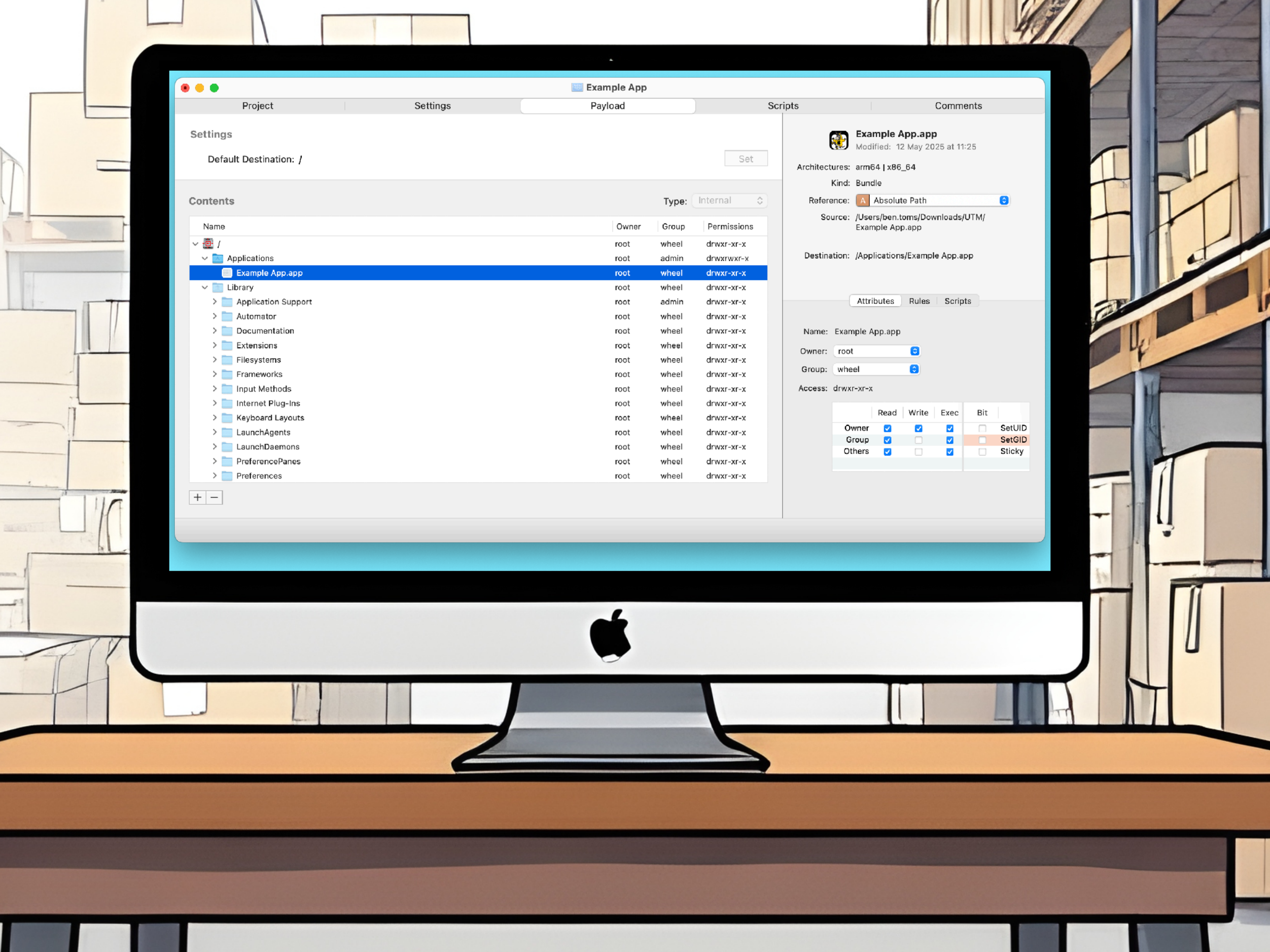

exec
PROJECT:
v1.x

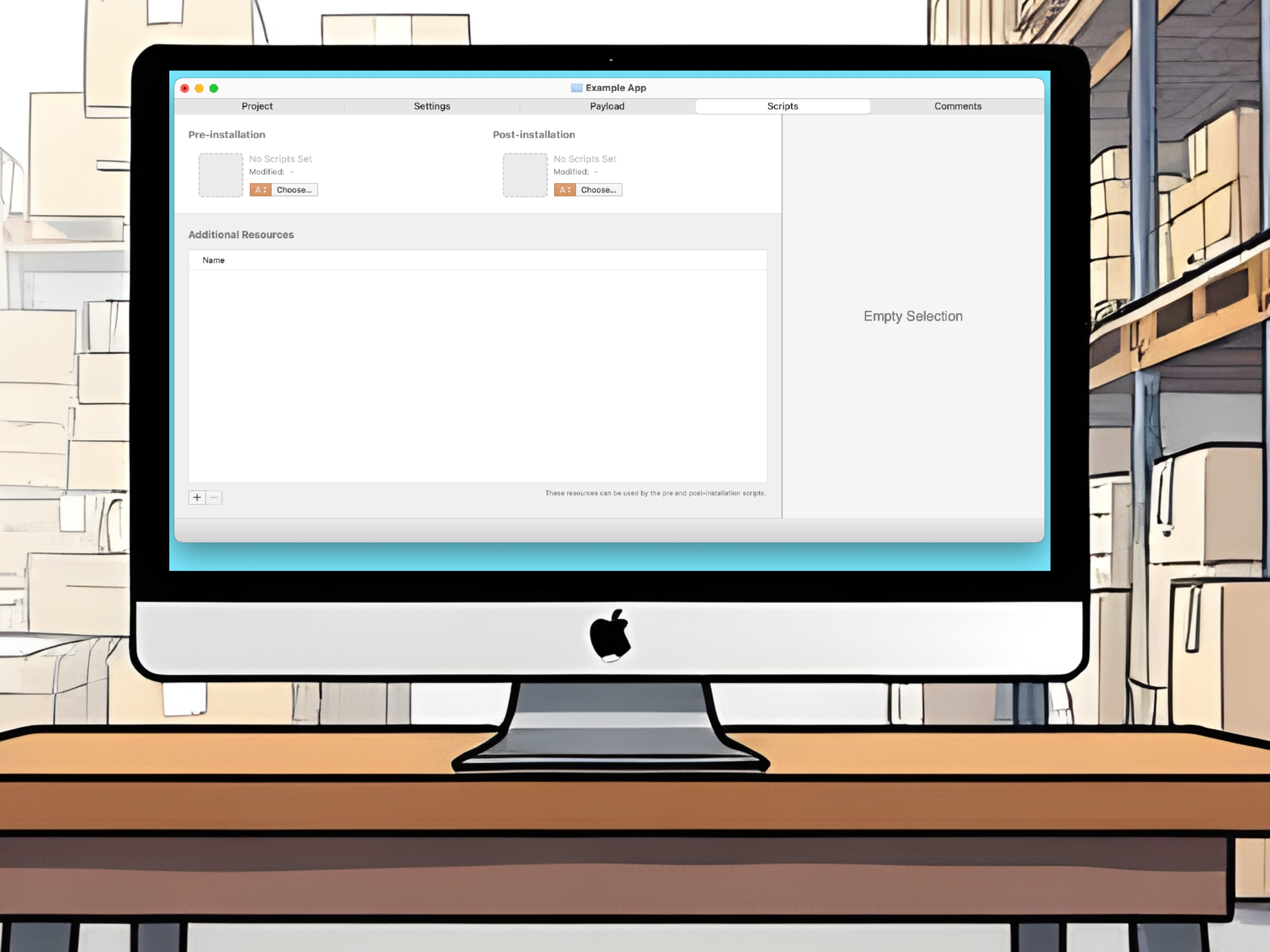
New Project
Choose a template for your project:
Distribution
Raw Package
Distribution
A Distribution project can include multiple packages and allows you to have a welcome message, custom background picture, etc.
Show on launch
Cancel
Previous
Next


New Project
Choose the name and location for your project:
Project Name:
Example App
Project Directory:
~/Desktop/Example App/
Choose...
The project directory ~/Desktop/Example App/ will be created if necessary, and the project file Example App.pkgproj will be created therein.
Show on launch
Cancel
Previous
Create


Example App
Project
Settings
Payload
Scripts
Comments
Build
Name:
Example App
Path:
R
build
Reference Folder:
Project Folder
Exclusions:
Pattern
Kind
RegEx
Remove .DS_Store files
Remove .pbdevelopment files
Remove SCM metadata
Optimize nib files
Remove Resources Disabled folders
Exclude items only from the payload


Example App
Project
Settings
Payload
Scripts
Comments
Tag
Identifier:
com.mygreatcompany.pkg.ExampleApp
Version:
1.0
Post-installation Behavior
On Success:
Do Nothing
Options
Require admin password for installation
Relocatable
Overwrite directory permissions
Follow symbolic links

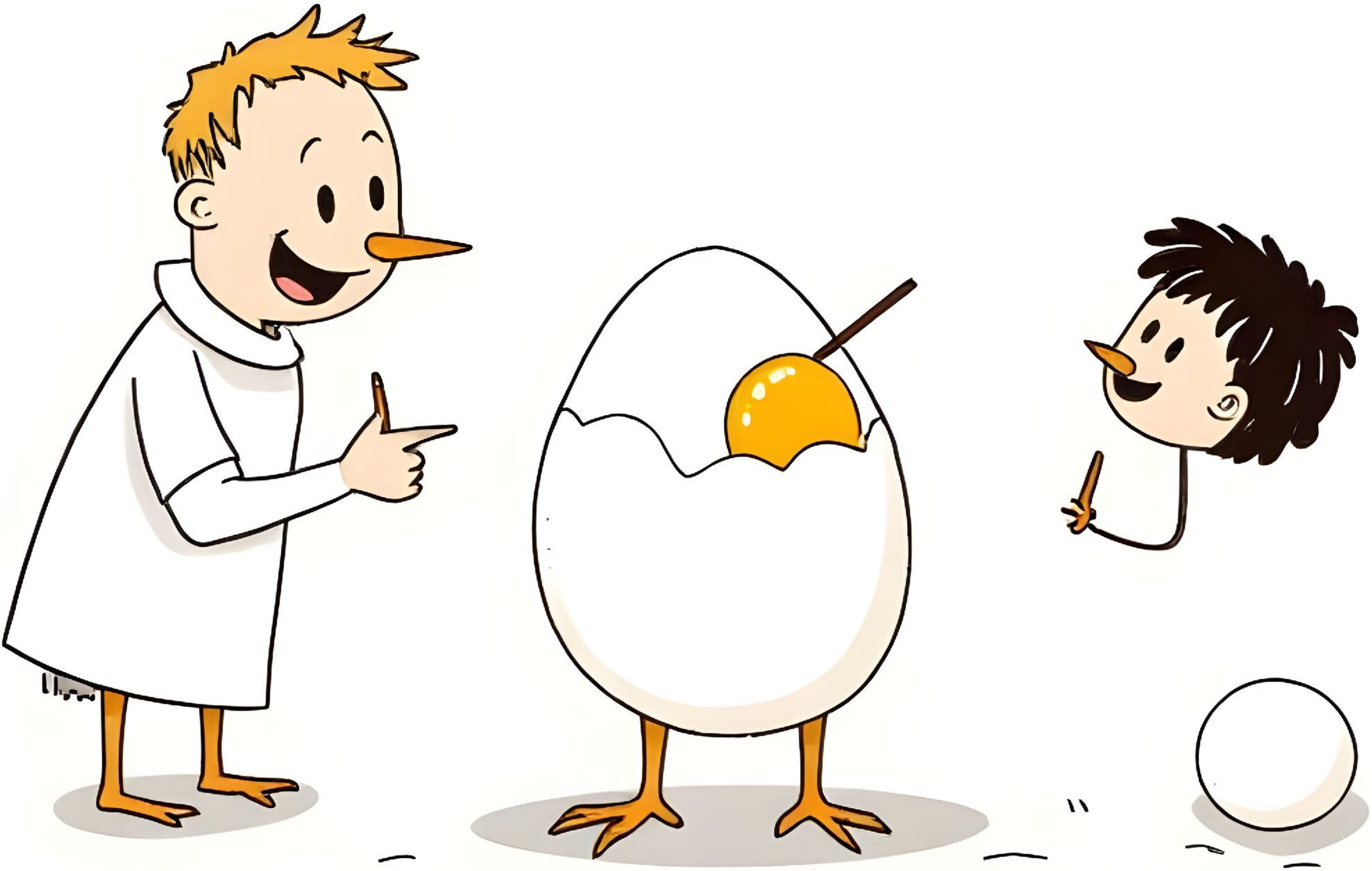

Example App
Project
Settings
Payload
Scripts
Comments
Settings
Default Destination: /
Set
Contents
Type: Internal
Name
Owner
Group
Permissions
/ root wheel drwxr-xr-x
Applications root admin drwxrwxr-x
Example App.app root wheel drwxr-xr-x
Library root wheel drwxr-xr-x
Application Support root admin drwxr-xr-x
Automator root wheel drwxr-xr-x
Documentation root wheel drwxr-xr-x
Extensions root wheel drwxr-xr-x
Filesystems root wheel drwxr-xr-x
Frameworks root wheel drwxr-xr-x
Input Methods root wheel drwxr-xr-x
Internet Plug-Ins root wheel drwxr-xr-x
Keyboard Layouts root wheel drwxr-xr-x
LaunchAgents root wheel drwxr-xr-x
LaunchDaemons root wheel drwxr-xr-x
PreferencePanes root wheel drwxr-xr-x
Preferences root wheel drwxr-xr-x
Example App.app
Modified: 12 May 2025 at 11:25
Architectures: arm64 | x86_64
Kind: Bundle
Reference: Absolute Path
Source: /Users/ben.toms/Downloads/UTM/Example App.app
Destination: /Applications/Example App.app
Attributes
Rules
Scripts
Name: Example App.app
Owner: root
Group: wheel
Access: drwxr-xr-x
Read
Write
Exec
Bit
Owner
Group
Others
SetUID
SetGID
Sticky

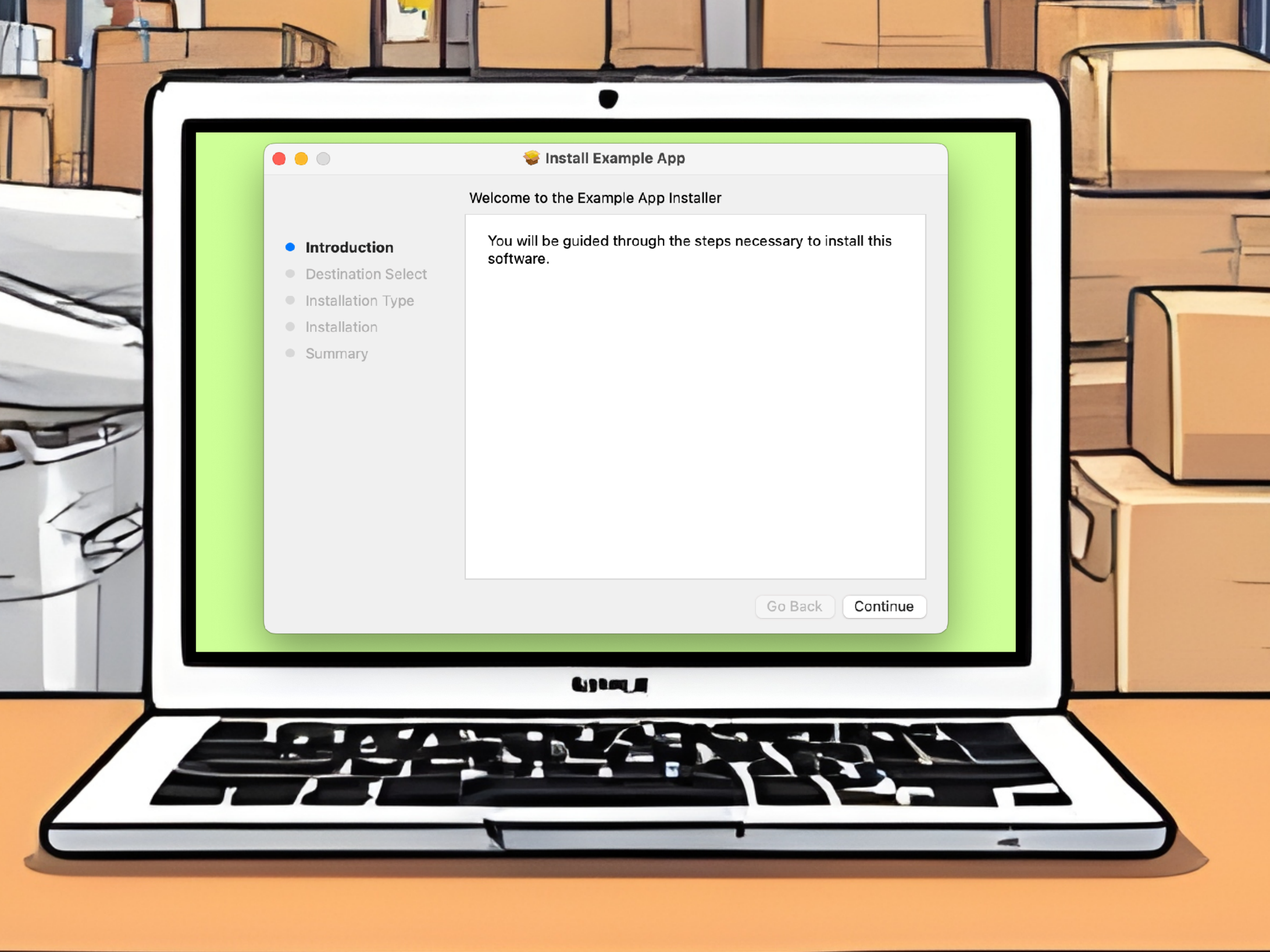

Example App
Project
Settings
Payload
Scripts
Comments
Pre-installation
No Scripts Set
Modified: -
Choose...
Post-installation
No Scripts Set
Modified: -
Choose...
Additional Resources
Name
These resources can be used by the pre and post-installation scripts.
Empty Selection

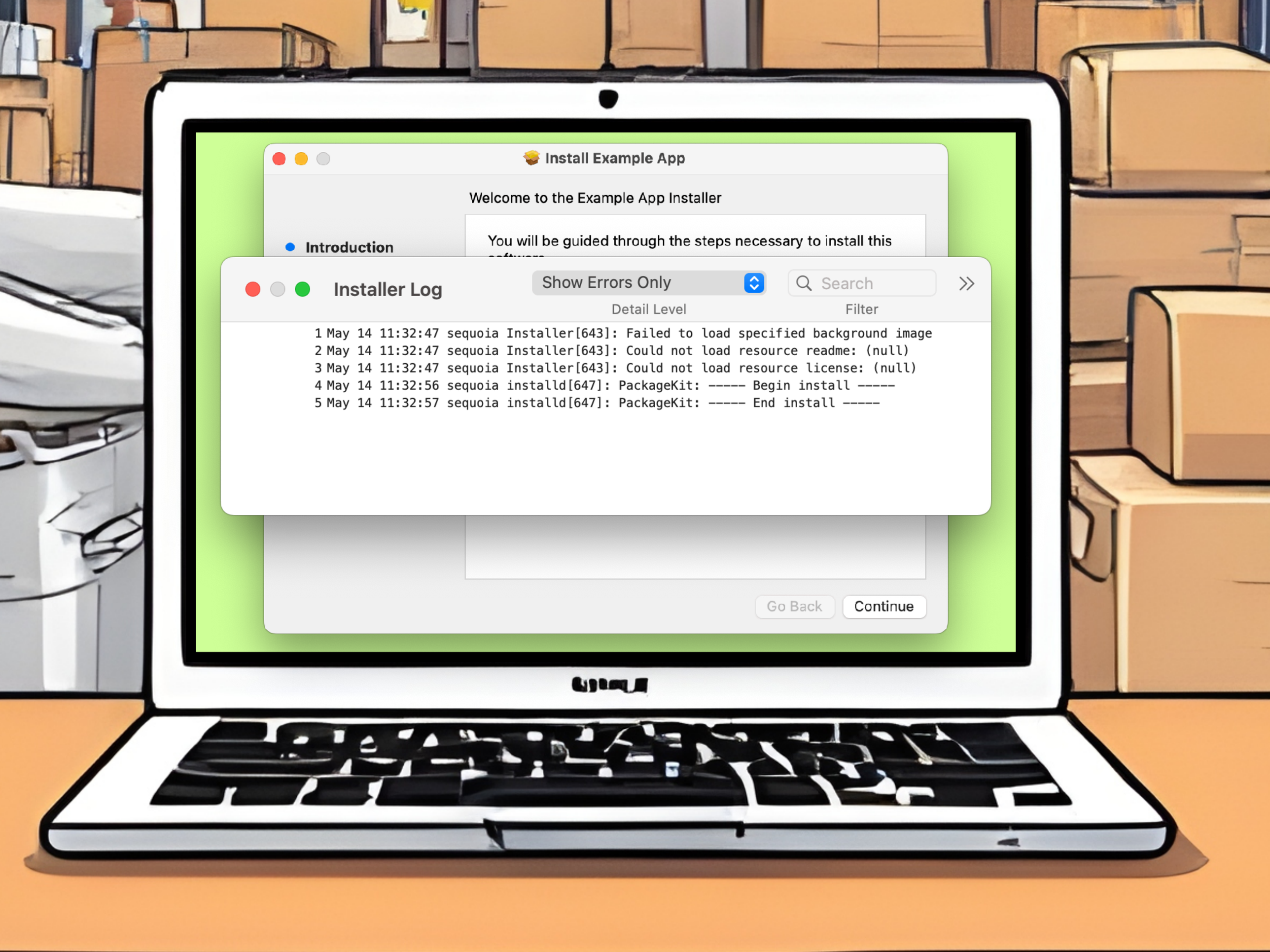
Example App
Project
Settings
Payload
Scripts
Comments

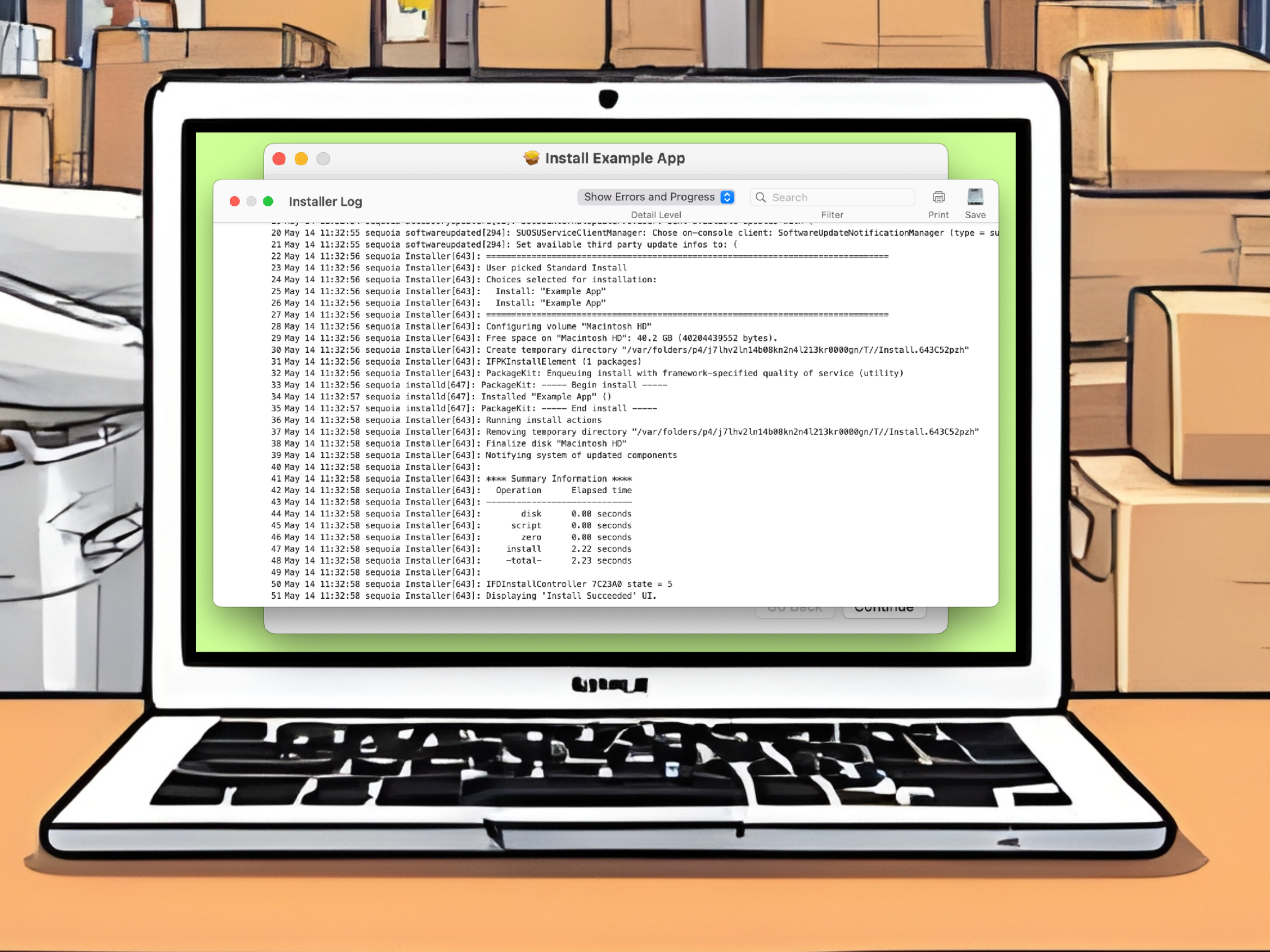
Open with Packages
PROJECT: PACKAGES
Example App
18 KB
Last modified on 14 May 2025 at 08:36:11

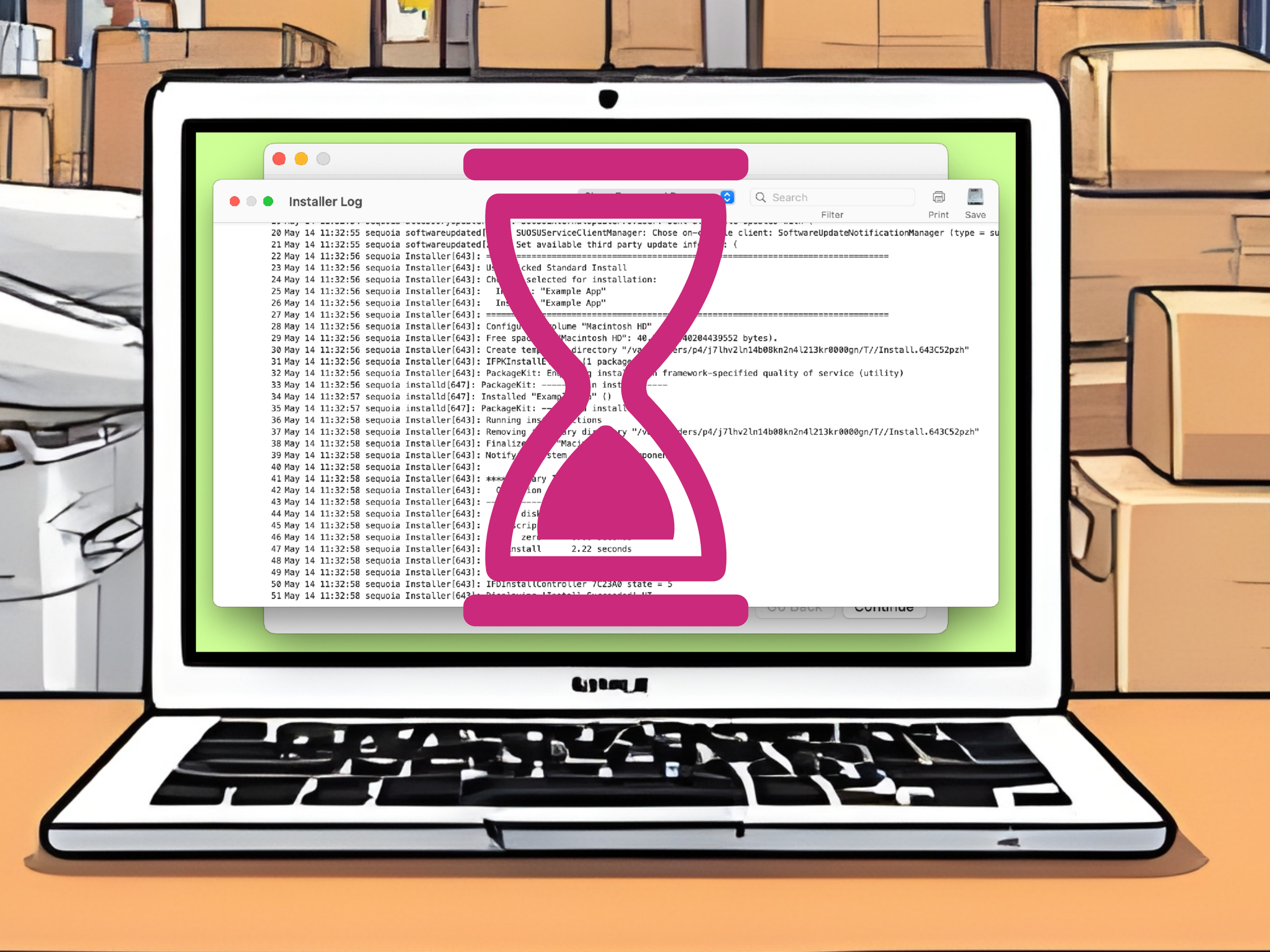
Installing packages

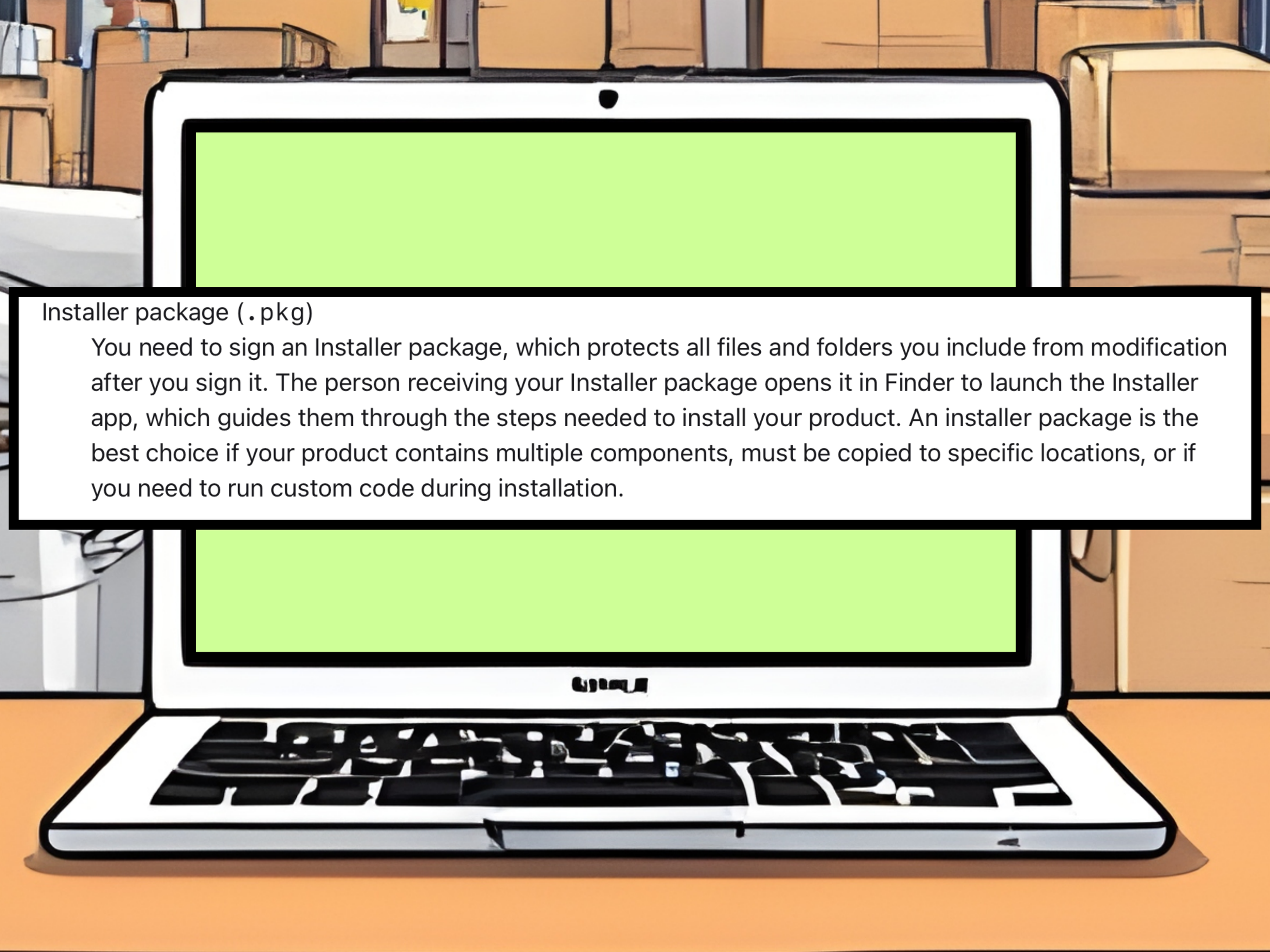



Install Example App
Welcome to the Example App Installer
Introduction
Destination Select
Installation Type
Installation
Summary
You will be guided through the steps necessary to install this software.
Go Back
Continue


Install Example App
Welcome to the Example App Installer
Introduction
You will be guided through the steps necessary to install this
Installer Log
Show Errors Only
Detail Level
Search
Filter
1 May 14 11:32:47 sequoia Installer[643]: Failed to load specified background image
2 May 14 11:32:47 sequoia Installer[643]: Could not load resource readme: (null)
3 May 14 11:32:47 sequoia Installer[643]: Could not load resource license: (null)
4 May 14 11:32:56 sequoia installd[647]: PackageKit: ----- Begin install -----
5 May 14 11:32:57 sequoia installd[647]: PackageKit: ----- End install -----
Go Back
Continue

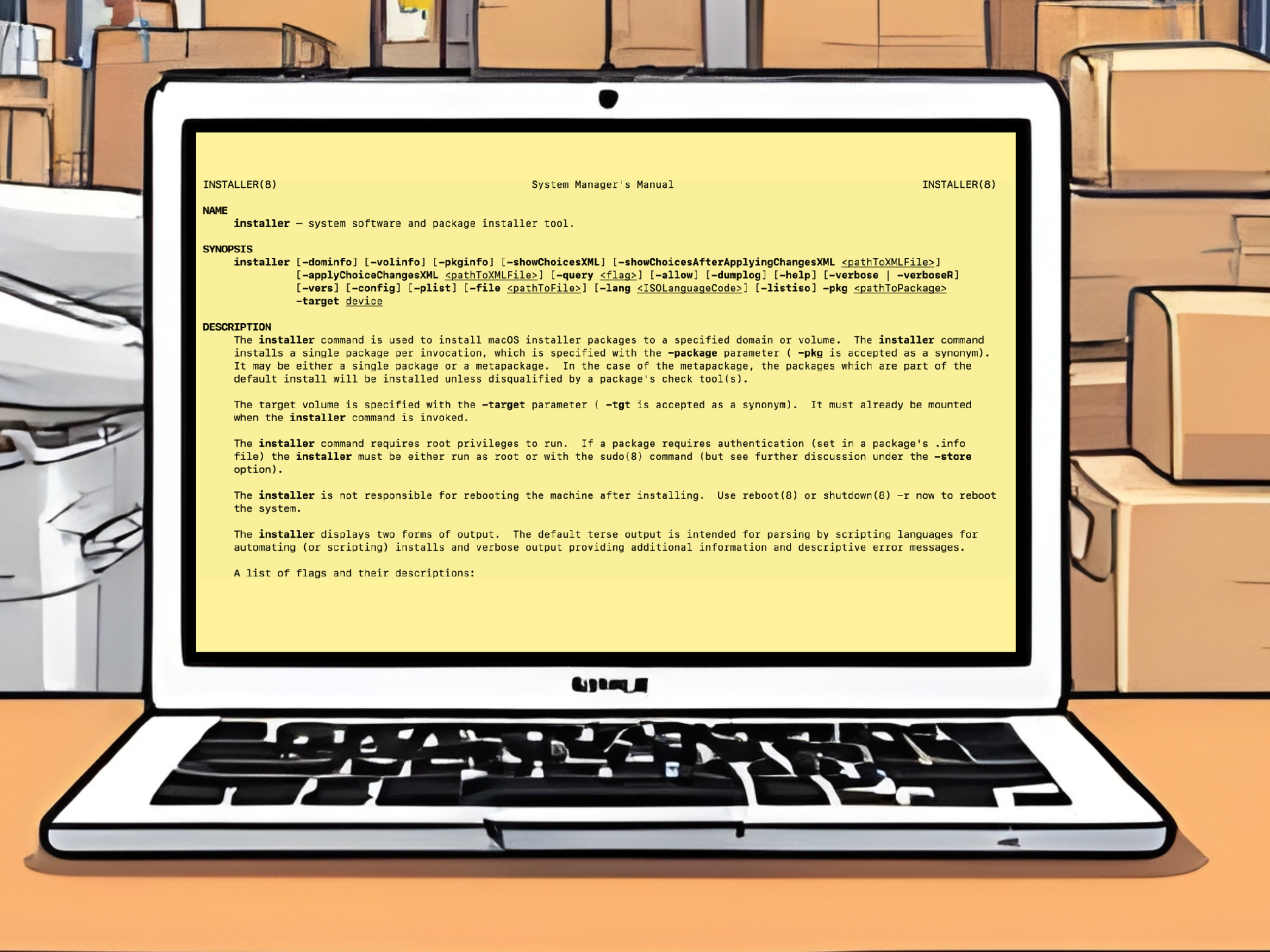

Install Example App
Installer Log
Show Errors and Progress
Search
Detail Level
Filter
Print
Save
20 May 14 11:32:55 sequoia softwareupdated[294]: SUOSUServiceClientManager: Chose on-console client: SoftwareUpdateNotificationManager (type = su
21 May 14 11:32:55 sequoia softwareupdated[294]: Set available third party update infos to: (
22 May 14 11:32:56 sequoia Installer[643]: ==========
23 May 14 11:32:56 sequoia Installer[643]: User picked Standard Install
24 May 14 11:32:56 sequoia Installer[643]: Choices selected for installation:
25 May 14 11:32:56 sequoia Installer[643]: Install: "Example App"
26 May 14 11:32:56 sequoia Installer[643]: Install: "Example App"
27 May 14 11:32:56 sequoia Installer[643]: ==========
28 May 14 11:32:56 sequoia Installer[643]: Configuring volume "Macintosh HD"
29 May 14 11:32:56 sequoia Installer[643]: Free space on "Macintosh HD": 40.2 GB (40204439552 bytes).
30 May 14 11:32:56 sequoia Installer[643]: Create temporary directory "/var/folders/p4/j7lhv2ln14b08kn2n4l213kr0000gn/T//Install.643C52pzh"
31 May 14 11:32:56 sequoia Installer[643]: IFPKInstallElement (1 packages)
32 May 14 11:32:56 sequoia Installer[643]: PackageKit: Enqueuing install with framework-specified quality of service (utility)
33 May 14 11:32:56 sequoia installd[647]: PackageKit: ----- Begin install -----
34 May 14 11:32:57 sequoia installd[647]: Installed "Example App" ()
35 May 14 11:32:57 sequoia installd[647]: PackageKit: ----- End install -----
36 May 14 11:32:58 sequoia Installer[643]: Running install actions
37 May 14 11:32:58 sequoia Installer[643]: Removing temporary directory "/var/folders/p4/j7lhv2ln14b08kn2n4l213kr0000gn/T//Install.643C52pzh"
38 May 14 11:32:58 sequoia Installer[643]: Finalize disk "Macintosh HD"
39 May 14 11:32:58 sequoia Installer[643]: Notifying system of updated components
40 May 14 11:32:58 sequoia Installer[643]:
41 May 14 11:32:58 sequoia Installer[643]: **** Summary Information ****
42 May 14 11:32:58 sequoia Installer[643]: Operation Elapsed time
43 May 14 11:32:58 sequoia Installer[643]: ----------
44 May 14 11:32:58 sequoia Installer[643]: disk 0.00 seconds
45 May 14 11:32:58 sequoia Installer[643]: script 0.00 seconds
46 May 14 11:32:58 sequoia Installer[643]: zero 0.00 seconds
47 May 14 11:32:58 sequoia Installer[643]: install 2.22 seconds
48 May 14 11:32:58 sequoia Installer[643]: -total- 2.23 seconds
49 May 14 11:32:58 sequoia Installer[643]:
50 May 14 11:32:58 sequoia Installer[643]: IFDInstallController 7C23A0 state = 5
51 May 14 11:32:58 sequoia Installer[643]: Displaying 'Install Succeeded' UI.


Installer Log
Search
Filter
Print
Save
20 May 14 11:32:55 sequoia softwareupdated[SUOSUServiceClientManager: Chose on-
21 May 14 11:32:55 sequoia softwareupdated[Set available third party update inf
22 May 14 11:32:56 sequoia Installer[643]:
23 May 14 11:32:56 sequoia Installer[643]: U cked Standard Install
24 May 14 11:32:56 sequoia Installer[643]: Ch selected for installation:
25 May 14 11:32:56 sequoia Installer[643]: I : "Example App"
26 May 14 11:32:56 sequoia Installer[643]: Ins "Example App"
27 May 14 11:32:56 sequoia Installer[643]:
28 May 14 11:32:56 sequoia Installer[643]: Configu olume "Macintosh HD"
29 May 14 11:32:56 sequoia Installer[643]: Free spac "Macintosh HD": 40 40204439552 bytes).
30 May 14 11:32:56 sequoia Installer[643]: Create tem directory "/va ers/p4/j7lhv2ln14b08kn2n4l213kr0000gn/T//Install.643C52pzh"
31 May 14 11:32:56 sequoia Installer[643]: IFPKInstallE (1 packag
32 May 14 11:32:56 sequoia Installer[643]: PackageKit: En g insta framework-specified quality of service (utility)
33 May 14 11:32:56 sequoia installd[647]: PackageKit: n inst
34 May 14 11:32:57 sequoia installd[647]: Installed "Examp " ()
35 May 14 11:32:57 sequoia installd[647]: PackageKit: -- install
36 May 14 11:32:58 sequoia Installer[643]: Running ins tions
37 May 14 11:32:58 sequoia Installer[643]: Removing ry di ry "/v ders/p4/j7lhv2ln14b08kn2n4l213kr0000gn/T//Install.643C52pzh"
38 May 14 11:32:58 sequoia Installer[643]: Finalize "Maci
39 May 14 11:32:58 sequoia Installer[643]: Notify stem pone
40 May 14 11:32:58 sequoia Installer[643]:
41 May 14 11:32:58 sequoia Installer[643]: *** ary
42 May 14 11:32:58 sequoia Installer[643]: ion
43 May 14 11:32:58 sequoia Installer[643]:
44 May 14 11:32:58 sequoia Installer[643]: dis
45 May 14 11:32:58 sequoia Installer[643]: scrip
46 May 14 11:32:58 sequoia Installer[643]: zer
47 May 14 11:32:58 sequoia Installer[643]: install 2.22 seconds
48 May 14 11:32:58 sequoia Installer[643]:
49 May 14 11:32:58 sequoia Installer[643]:
50 May 14 11:32:58 sequoia Installer[643]: IFDInstallController 7C23A0 state = 5
51 May 14 11:32:58 sequoia Installer[643]:



Installer package (`.pkg`)

You need to sign an Installer package, which protects all files and folders you include from modification after you sign it. The person receiving your Installer package opens it in Finder to launch the Installer app, which guides them through the steps needed to install your product. An installer package is the best choice if your product contains multiple components, must be copied to specific locations, or if you need to run custom code during installation.



INSTALLER(8) System Manager's Manual INSTALLER(8)

NAME

installer — system software and package installer tool.

SYNOPSIS

installer [**-dominfo**] [**-volinfo**] [**-pkginfo**] [**-showChoicesXML**] [**-showChoicesAfterApplyingChangesXML** <pathToXMLFile>] [**-applyChoiceChangesXML** <pathToXMLFile>] [**-query** <flag>] [**-allow**] [**-dumplog**] [**-help**] [**-verbose** | **-verboseR**] [**-vers**] [**-config**] [**-plist**] [**-file** <pathToFile>] [**-lang** <ISOLanguageCode>] [**-listiso**] **-pkg** <pathToPackage> **-target** device

DESCRIPTION

The **installer** command is used to install macOS installer packages to a specified domain or volume. The **installer** command installs a single package per invocation, which is specified with the **-package** parameter (**-pkg** is accepted as a synonym). It may be either a single package or a metapackage. In the case of the metapackage, the packages which are part of the default install will be installed unless disqualified by a package's check tool(s).

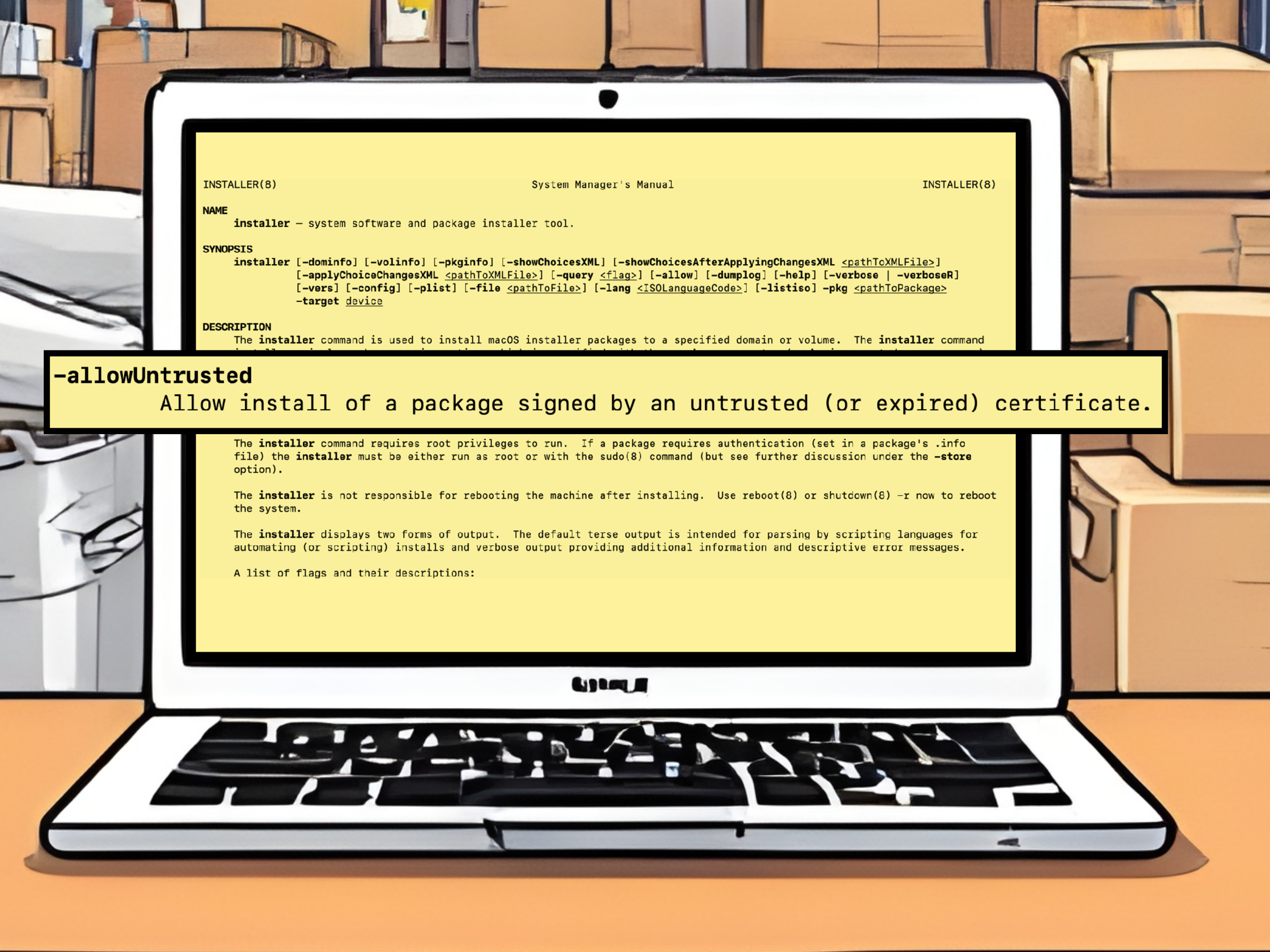
The target volume is specified with the **-target** parameter (**-tgt** is accepted as a synonym). It must already be mounted when the **installer** command is invoked.

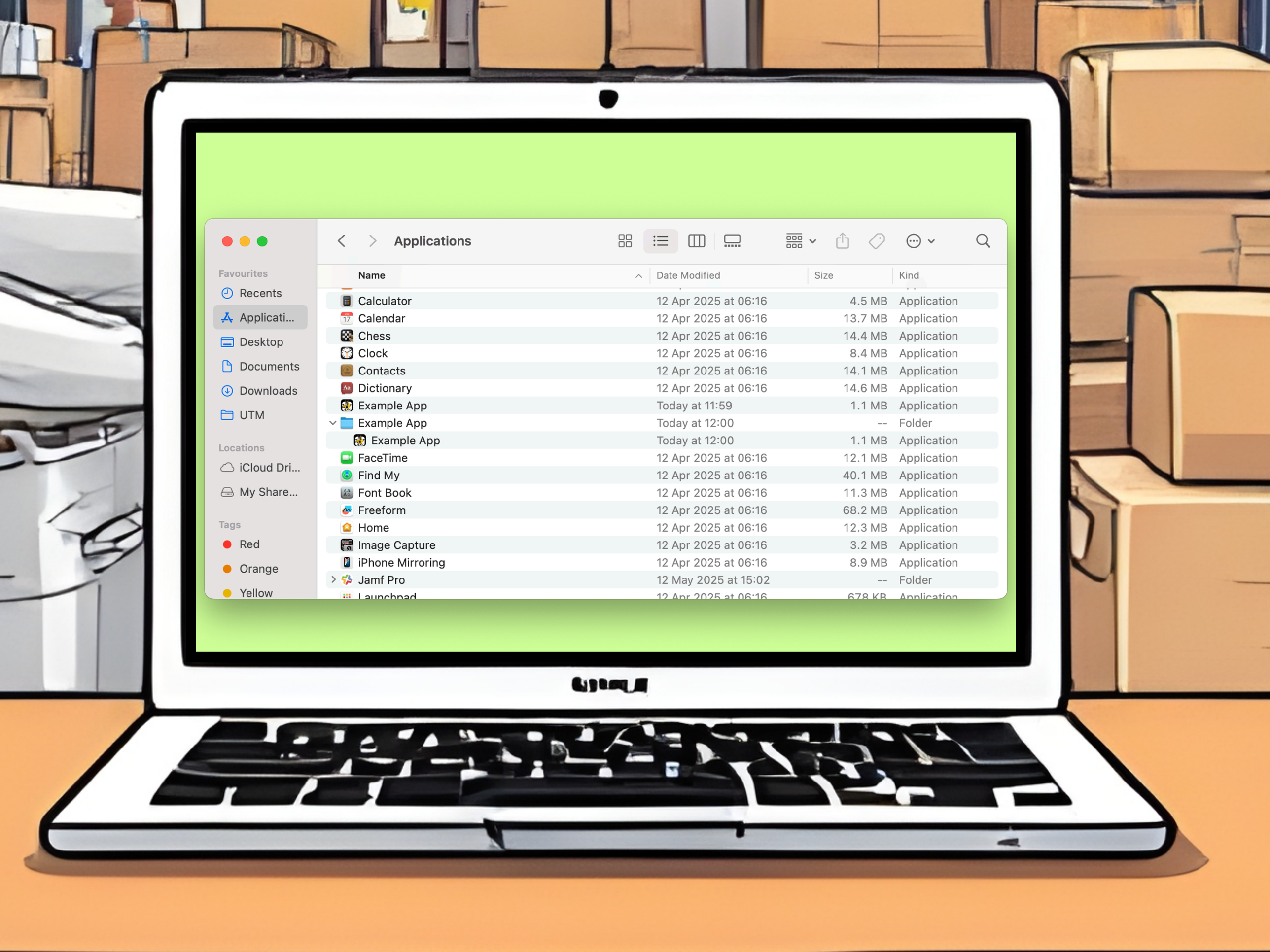
The **installer** command requires root privileges to run. If a package requires authentication (set in a package's .info file) the **installer** must be either run as root or with the sudo(8) command (but see further discussion under the **-store** option).

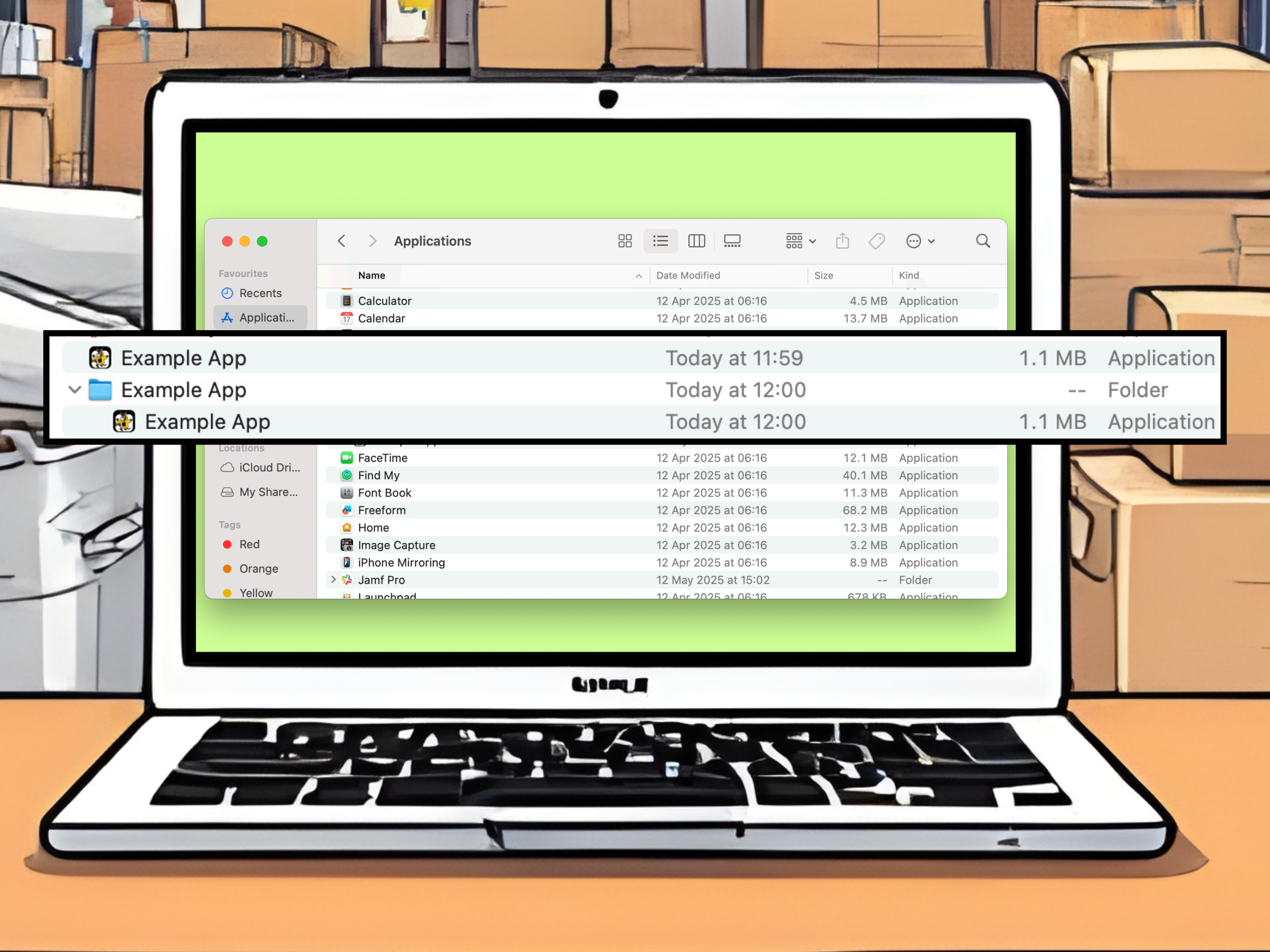
The **installer** is not responsible for rebooting the machine after installing. Use reboot(8) or shutdown(8) -r now to reboot the system.

The **installer** displays two forms of output. The default terse output is intended for parsing by scripting languages for automating (or scripting) installs and verbose output providing additional information and descriptive error messages.

A list of flags and their descriptions:


INSTALLER(8) System Manager's Manual INSTALLER(8)
NAME
installer — system software and package installer tool.
SYNOPSIS
installer [-dominfo] [-volinfo] [-pkginfo] [-showChoicesXML] [-showChoicesAfterApplyingChangesXML <pathToXMLFile>]
[-applyChoiceChangesXML <pathToXMLFile>] [-query <flag>] [-allow] [-dumplog] [-help] [-verbose | -verboseR]
[-vers] [-config] [-plist] [-file <pathToFile>] [-lang <ISOLanguageCode>] [-listiso] -pkg <pathToPackage>
-target device
DESCRIPTION
The installer command is used to install macOS installer packages to a specified domain or volume. The installer command
-allowUntrusted
Allow install of a package signed by an untrusted (or expired) certificate.
The installer command requires root privileges to run. If a package requires authentication (set in a package's .info file) the installer must be either run as root or with the sudo(8) command (but see further discussion under the -store option).
The installer is not responsible for rebooting the machine after installing. Use reboot(8) or shutdown(8) -r now to reboot the system.
The installer displays two forms of output. The default terse output is intended for parsing by scripting languages for automating (or scripting) installs and verbose output providing additional information and descriptive error messages.
A list of flags and their descriptions:


Applications
Favourites
Recents
Applicati...
Desktop
Documents
Downloads
UTM
Locations
iCloud Dri...
My Share...
Tags
Red
Orange
Yellow
Name
Date Modified
Size
Kind
Calculator 12 Apr 2025 at 06:16 4.5 MB Application
Calendar 12 Apr 2025 at 06:16 13.7 MB Application
Chess 12 Apr 2025 at 06:16 14.4 MB Application
Clock 12 Apr 2025 at 06:16 8.4 MB Application
Contacts 12 Apr 2025 at 06:16 14.1 MB Application
Dictionary 12 Apr 2025 at 06:16 14.6 MB Application
Example App Today at 11:59 1.1 MB Application
Example App Today at 12:00 -- Folder
Example App Today at 12:00 1.1 MB Application
FaceTime 12 Apr 2025 at 06:16 12.1 MB Application
Find My 12 Apr 2025 at 06:16 40.1 MB Application
Font Book 12 Apr 2025 at 06:16 11.3 MB Application
Freeform 12 Apr 2025 at 06:16 68.2 MB Application
Home 12 Apr 2025 at 06:16 12.3 MB Application
Image Capture 12 Apr 2025 at 06:16 3.2 MB Application
iPhone Mirroring 12 Apr 2025 at 06:16 8.9 MB Application
Jamf Pro 12 May 2025 at 15:02 -- Folder


Applications
Favourites
Recents
Applicati...
Name
Date Modified
Size
Kind
Calculator 12 Apr 2025 at 06:16 4.5 MB Application
Calendar 12 Apr 2025 at 06:16 13.7 MB Application
Example App Today at 11:59 1.1 MB Application
Example App Today at 12:00 -- Folder
Example App Today at 12:00 1.1 MB Application
Locations
iCloud Dri...
My Share...
Tags
Red
Orange
Yellow
FaceTime 12 Apr 2025 at 06:16 12.1 MB Application
Find My 12 Apr 2025 at 06:16 40.1 MB Application
Font Book 12 Apr 2025 at 06:16 11.3 MB Application
Freeform 12 Apr 2025 at 06:16 68.2 MB Application
Home 12 Apr 2025 at 06:16 12.3 MB Application
Image Capture 12 Apr 2025 at 06:16 3.2 MB Application
iPhone Mirroring 12 Apr 2025 at 06:16 8.9 MB Application
Jamf Pro 12 May 2025 at 15:02 -- Folder

Uninstalling packages

pkgutil(1) General Commands Manual pkgutil(1)

NAME

pkgutil — Query and manipulate macOS Installer packages and receipts.

SYNOPSIS

pkgutil [options] [commands]

DESCRIPTION

pkgutil reads and manipulates macOS Installer flat packages, and provides access to the “receipt” database used by the Installer. Options are processed first, and affect the operation of all commands. Multiple commands are performed sequentially in the given order.

The files and directories where receipts are stored are subject to change. Always use pkgutil to query or modify them.

OPTIONS

--help, -h
A brief summary of commands and usage.

--force, -f
Don't ask for confirmation before performing a potentially destructive or ambiguous operation.

--verbose, -v
Output in a "human-readable" format with extra headers, footers, indentation, and other contextual information.

--volume path
Perform all operations on the specified volume or home directory. The root volume '/' will be used if unspecified.

--edit-pkg package-id
Specifies an existing receipt to be modified in-place by **--learn**.

--only-files

/usr/sbin/pkgutil --pkgs

com.apple.files.data-template
com.apple.pkg.GatekeeperCompatibilityData.16U1906
com.apple.pkg.XProtectPlistConfigData_10_15.16U4368
com.apple.pkg.MRTConfigData_10_15.16U4211
com.apple.pkg.XProtectPlistConfigData_10_15.16U4367
com.apple.pkg.XProtectPayloads_10_15.16U4357
com.macmule.ExampleApp

/usr/sbin/pkgutil --files
com.macmule.ExampleApp

```
._Example App.app
Example App.app
Example App.app/._Icon
Example App.app/Contents
Example App.app/Contents/Info.plist
Example App.app/Contents/MacOS
Example App.app/Contents/MacOS/applet
Example App.app/Contents/PkgInfo
Example App.app/Contents/Resources
Example App.app/Contents/Resources/._description.rtfd
Example App.app/Contents/Resources/Scripts
Example App.app/Contents/Resources/Scripts/main.scpt
Example App.app/Contents/Resources/applet.icns
Example App.app/Contents/Resources/applet.rsrc
Example App.app/Contents/Resources/description.rtfd
Example App.app/Contents/Resources/description.rtfd/TXT.rtf
Example App.app/Contents/_CodeSignature
Example App.app/Contents/_CodeSignature/._CodeResources
Example App.app/Contents/_CodeSignature/CodeResources
```

--files package-id
List all of the files installed under the package-id.

--export-plist package-id
Print all receipt information about the specified package-id in the standard Mac OS X plist(5) format.

--pkg-info package-id
Print extended information about the specified package-id.

--pkg-info-plist package-id
Print extended information about the specified package-id in Mac OS X plist(5) format.

--forget package-id
Discard all receipt data about package-id, but do not touch the installed files. **DO NOT** use this command from an installer package script to fix broken package design.

--learn path
Update the ACLs of the given path in the receipt identified by **--edit-pkg**. This affects subsequent repair operations on the package. This command cannot be used from package postinstall scripts, but if a postinstall script changes the ACLs on the installed files, the receipt is automatically be updated to reflect those changes. This command will not update the filesystem permissions in the receipt.

--pkg-groups package-id
List all of the package groups this package-id is a member of.

--groups
List all of the package groups on the specified **--volume**.

--groups-plist
List all of the package groups on the specified **--volume** in Mac OS X plist(5) format.

:

--files package-id
List all of the files installed under the package-id.

--export-plist package-id
Print all receipt information about the specified package-id in the standard Mac OS X plist(5) format.

--pkg-info package-id
Print extended information about the specified package-id.

--pkg-info-plist package-id
Print extended information about the specified package-id in Mac OS X plist(5) format.

--forget package-id
Discard all receipt data about package-id, but do not touch the installed files. **DO NOT** use this command from an installer package script to fix broken package design.

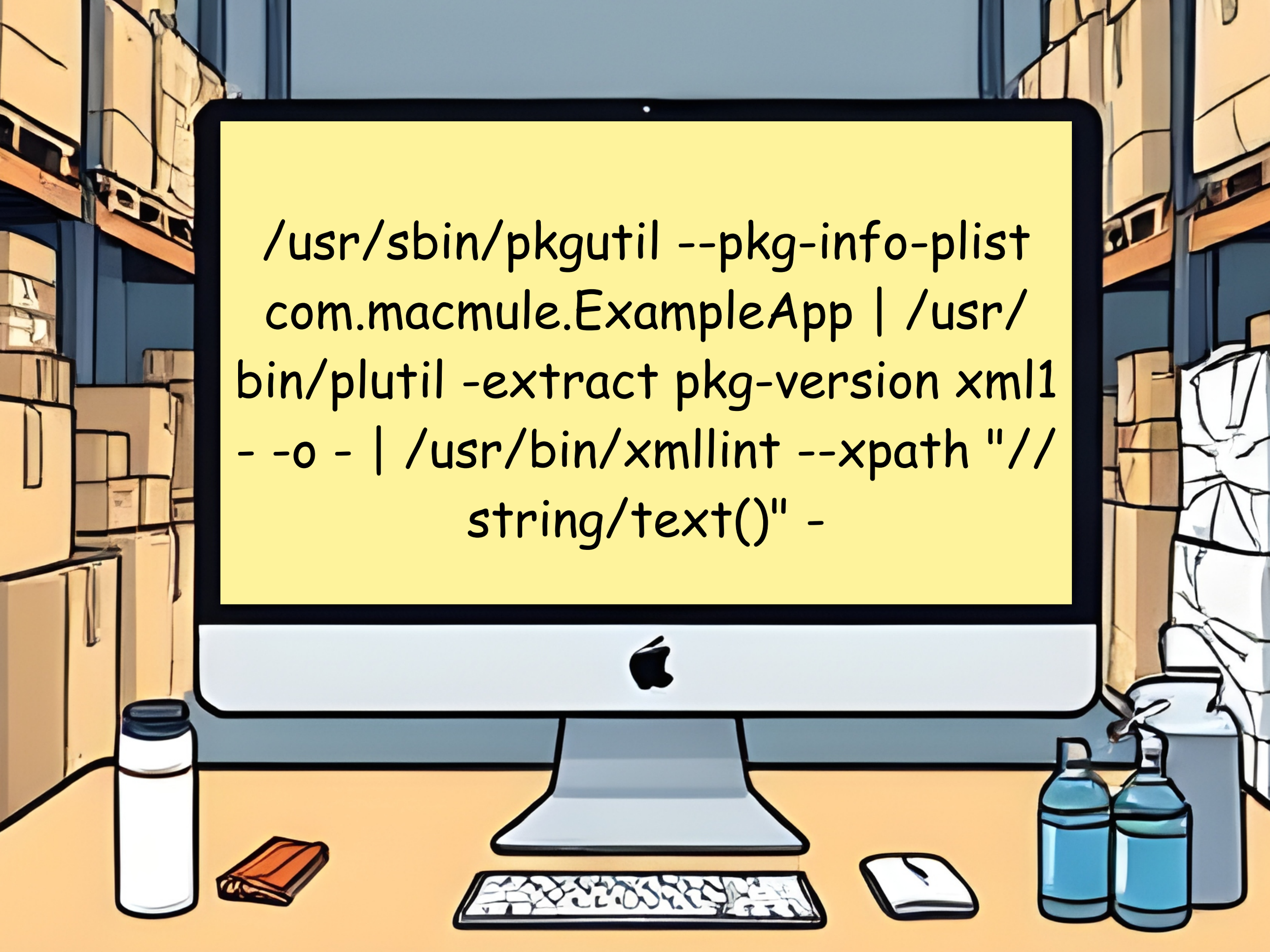
script changes the ACLs on the installed files, the receipt is automatically be updated to reflect those changes. This command will not update the filesystem permissions in the receipt.

--pkg-groups package-id
List all of the package groups this package-id is a member of.

--groups
List all of the package groups on the specified **--volume**.

--groups-plist
List all of the package groups on the specified **--volume** in Mac OS X plist(5) format.

:


/usr/sbin/pkgutil --pkg-info-plist com.macmule.ExampleApp | /usr/bin/plutil -extract pkg-version xml1 - -o - | /usr/bin/xmllint --xpath "//string/text()" -

1

1
<key>CFBundleShortVersionString</key>
<string>1</string>
Info.plist

1

```
<key>CFBundleShortVersionString</key>
<string>1</string>
```

Info.plist

```
<pkg-ref id="com.macmule.ExampleApp.payload.pkg" installKBytes="176" version="1" auth="Root">#payload.pkg</pkg-ref>
```

Distribution

Wrapping up...

Search

Apple Books

- Home
- Book Store
- Audiobook Store

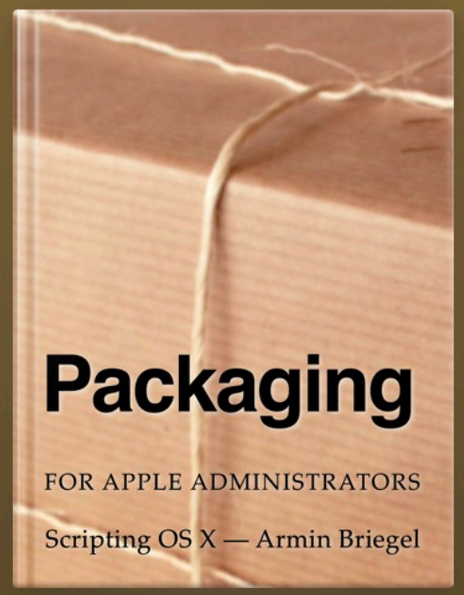
Library

- All
- Want to Read
- Finished
- Books
- Audiobooks
- PDFs
- My Samples

My Collections

- My Books
- New Collection

B Ben Toms



Packaging for Apple Administrators

Armin Briegel >

★ 5.0 (1) • Computing & Internet

Book ⓘ
117 Pages

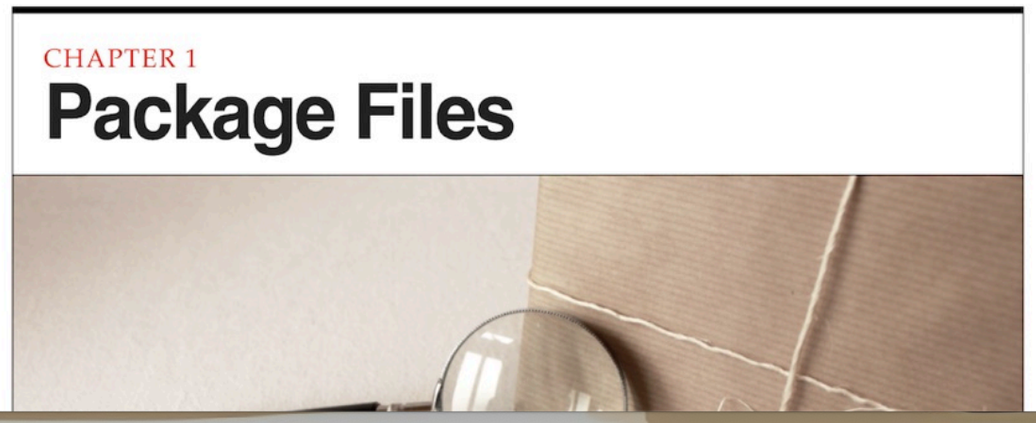
Download

From the Publisher

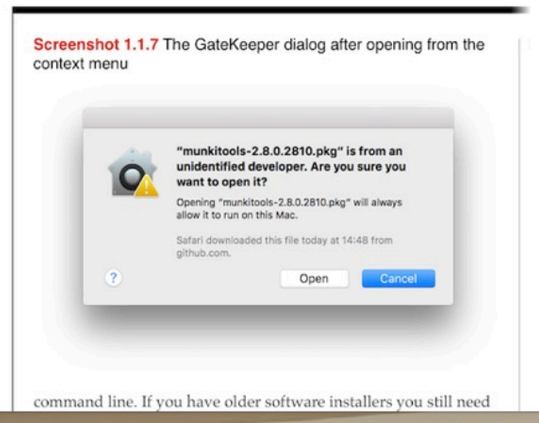
New, updated version!

Oct 14, 2019 with new examples and Catalin... More**

From the Book


CHAPTER 1

Package Files


Screenshot 1.1.7 The GateKeeper dialog after opening from the context menu

"munkitools-2.8.0.2810.pkg" is from an unidentified developer. Are you sure you want to open it?

Opening "munkitools-2.8.0.2810.pkg" will always allow it to run on this Mac.

Safari downloaded this file today at 14:48 from github.com.

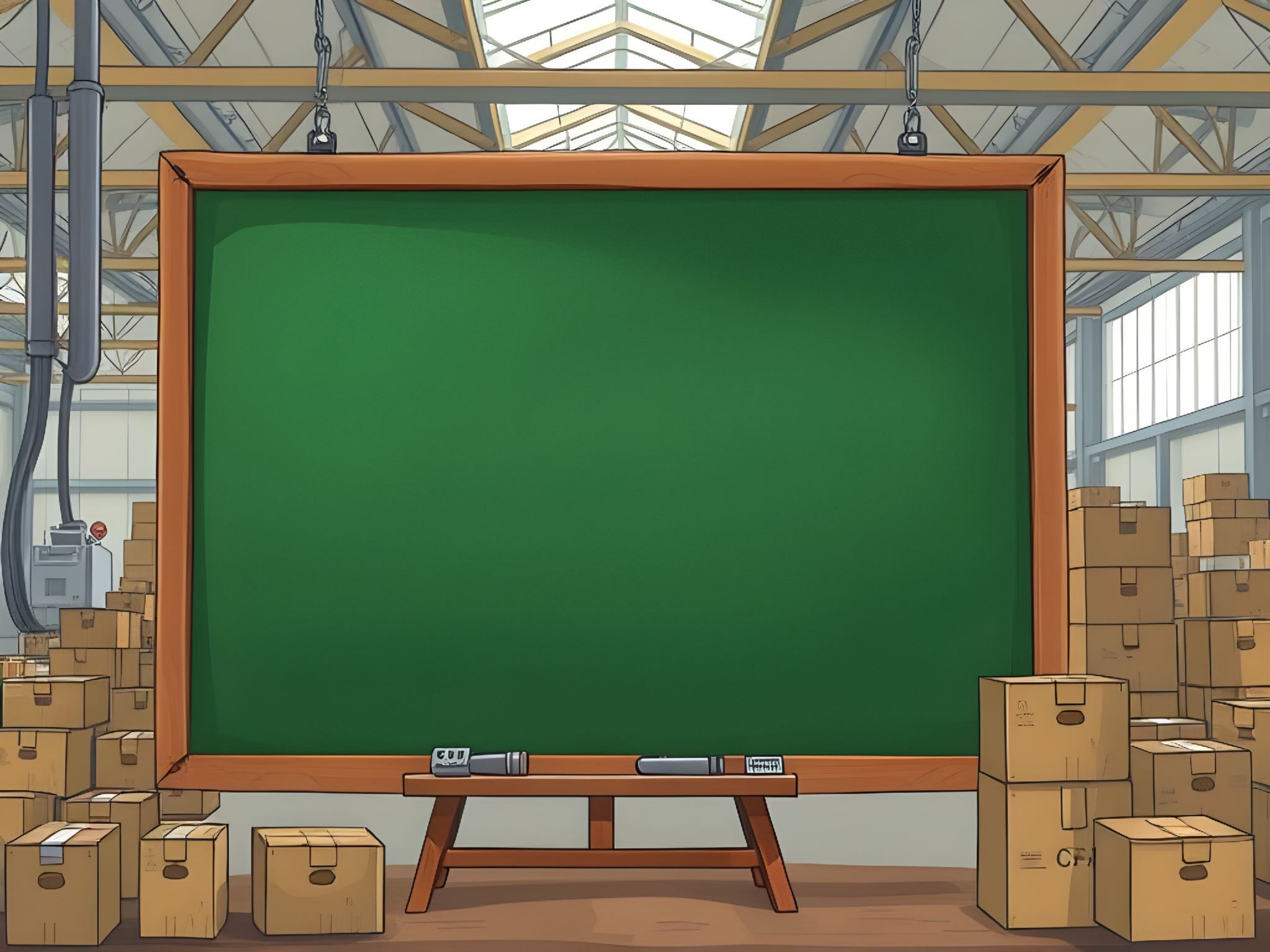
Open Cancel

command line. If you have older software installers you still need

#autopkg

#packaging

Thank you!!



10th July 2025 - Meet Up @ Jamf

May 12, 2025

For our July meet up of 2025, the great folks at Jamf are hosting us again with food and beverages aplenty.

The schedule is as follows:

- 18:30 - 19:00: Arrive and mingle
- 19:00 - 19:35: “Declarative management of Jamf Pro with terraform” - Dafydd Watkins & Joe Little - Lloyds Banking Group
- 19:40 - 20:00: “Turning not-HomeKit stuff into kinda-HomeKit stuff with Homebridge” - Ben Toms - Jamf
- 20:05 - 20:25: TBD
- 20:30 - 20:50: TBD