



# Apple Scripting with Jamf 201:

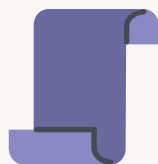
An intermediate guide to automating common tasks

If you're an Apple admin who would like to take the basic scripting skills you learned in [Scripting 101](#) to the next level, this guide is for you!

**You can automate three common IT tasks when managing Apple devices with three simple, powerful tools for scripting:**



**Writing text to a file  
and retrieving it for  
later use**



**Using script parameters  
in Jamf Pro**



**Creating interactive  
dialogs using jamfHelper  
and osascript**

## Write and read a file

Sometimes, you need to leave information on a Mac so that you can access it later. For example, you may wish to leave behind a provisioning receipt that lets you know when a Mac was prepared and deployed and who built it.

Here are a few ways to do just that:

### Through Terminal

Remember the “echo” command from our Scripting 101 webinar? This command echoes, or prints, whatever you input. If you input:

```
echo 'Hello, World!'
```

...and press return, it outputs:

```
Hello, World!
```

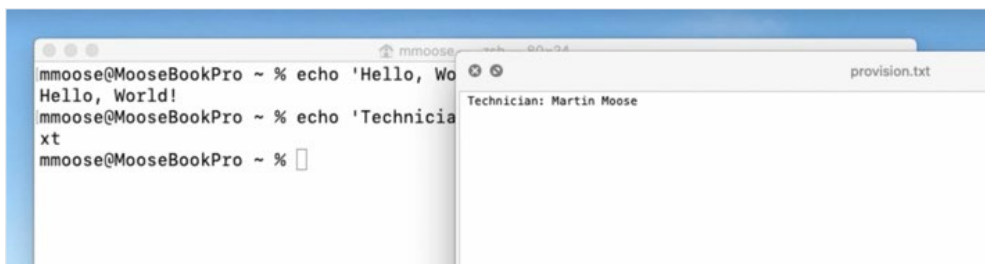
### Saving to a text file with the echo command

By using the >> sign, which is called the “redirection” symbol, you can save to a file. If the file doesn’t exist, the redirection symbol will create it. If it already exists, it’ll append whatever you’re echoing to the end of the file. (If you use just one redirection, >, it overwrites the contents of the file.)

```
echo Technician: Martin Moose >> ~/Desktop/provision.txt
echo "Date: $( /bin/date +%y-%m-%d )" >> !$
echo Department: Graphics >> !$
```

When you press return, the file appears on your Desktop.

You can select the file and tap spacebar to view it in QuickLook.



## Adding the current date to a file

To automate adding the current date to a file, use the command:

```
echo "Date: $( /bin/date +%y-%m-%d )" >
```

The '%y-%m-%d' stand for year, month and date.

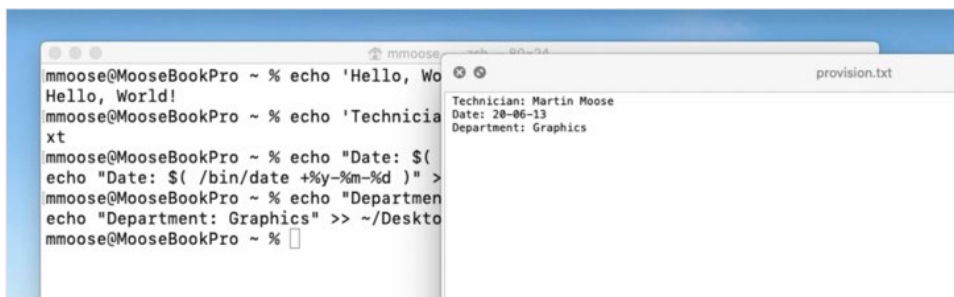
To save yourself a couple of keystrokes, use a shortcut which repeats the last argument of the previous command to redirect the text into the file:

```
>> !$
```

And then write a department name to the file:

```
echo Department: Graphics >> !$
```

When we QuickLook the file now, all of our text is in there in the order we added it:



To read it out again, use the "cat" command followed by the path to the file:

```
/bin/cat ~/Desktop/provision.txt
```

It will then display the contents of that file directly in Terminal:

```
Technician: Martin Moose  
Date: 20-06-13  
Department: Graphics
```

## Review

<code>echo</code>	Prints what you input into the Terminal window
<code>&gt;&gt;</code>	Redirects output to a file/append existing file
<code>&gt;</code>	Redirects output to a file/overwrite existing file
<code>\$( command )</code>	Runs a command
<code>!\$</code>	Repeats the last argument
<code>cat</code>	Reads a file

## Defaults Command

The “cat” command lets you read the entire file, but what if you want just one piece of information from it?

The ‘defaults’ command is the same command you would use to read plists in your preferences folder, and can also be used to read back information.

We can also use it to write our own plists.

To start a new file, use “defaults write” and tell it where you want the file. After that, provide a one-word description “Technician” to indicate the technician who built this computer and follow it with the name of the technician.

```
/usr/bin/defaults write ~/Desktop/provision.plist Technician 'Martin Moose'
```

Press return.

The file appears on the desktop, and when you use QuickLook to view it, it’s very different from the first one:

```
<?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>Technician</key>
    <string>Martin Moose</string>
  </dict>
</plist>
```

The “defaults” command adds a lot of formatting, but if you look closely, the information is all there.

Let’s add the date and the department and look at the file again:

```
/usr/bin/defaults write ~/Desktop/provision.plist Technician 'Martin Moose'
/usr/bin/defaults write ~/Desktop/provision.plist Date $( /bin/date '+%y-%m-%
d' ) /usr/bin/defaults write ~/Desktop/provision.plist Department 'Graphics'
```

Now, the file shows all three pieces of information:

```
<?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>Date</key>
    <string>20-06-13</string>
    <key>Department</key>
    <string>Graphics</string>
    <key>Technician</key>
    <string>Martin Moose</string>
  </dict>
</plist>
```

Because this is a plist, each type of information —Technician, Date and Department— is listed as a "key" and the value for that type of information is listed below each key. (The defaults command automatically alphabetizes keys. No matter what order you add them, they'll be listed alphabetically.)

The plist looks a lot more complex than our first text file, but here's where it shines:

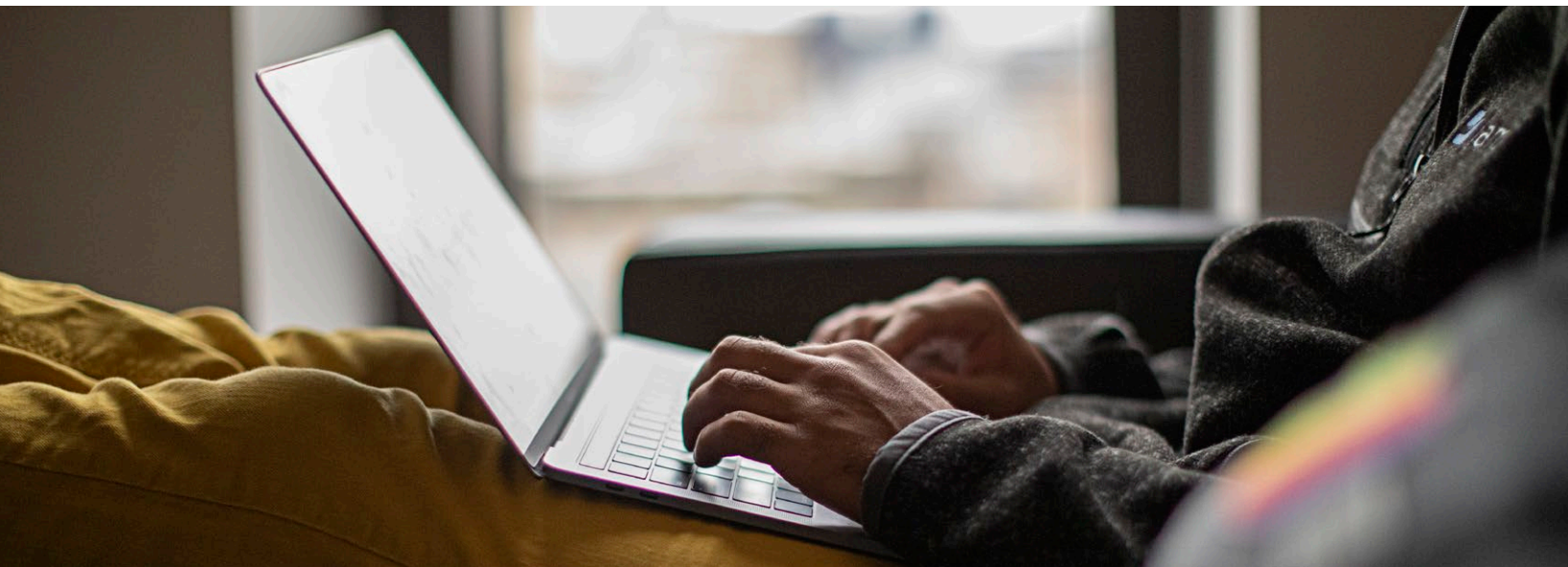
Use "defaults read" to read the file, and then specify "Technician," "Date" or "Department" to return the value of just one item. This is particularly useful for something like an Extension Attribute where an admin would use this command in a short script to echo the result back to Jamf Pro.

Example:

```
/usr/bin/defaults read~/Desktop/provision.plist Date
```

The Terminal will deliver only the date:

```
20-06-13
```



# Using script parameters in Jamf Pro

## What are script parameters?

Let's demonstrate with a quick script.

```
#!/bin/zsh
```

The shebang is zsh or zee-shell, like "seashell."

I'll add an echo statement, which will just give me an empty line to make things easier to see:

```
echo
```

Then I'll add another echo statement with parameter variables "1" through "5."

```
echo $1 $2 $3 $4 $5
```

Remember, when something in a script starts with a "\$" that's usually an indicator that something is a variable or placeholder for something.

The full script:

```
#!/bin/zsh
echo
echo $1 $2 $3 $4 $5
```

Save the script to the desktop as parameters.zsh.

Any time you create a new script file, you have to use a command in Terminal called "chmod" to make it executable. Otherwise, Terminal thinks it's just a plain text file instead of a script that it can run.

Type in the word "chmod," which means "change mode," and then add +x, which means make it "executable," and drag your script into the window.

Now, when you run the script followed by "Mary had a little lamb," it echoes back "Mary had a little lamb:"

```
chmod +x ~/Desktop/parameters.zsh Mary had a little lamb
Mary had a little lamb
```



Use your favorite code editor or any plain text editor like BBEdit or even TextEdit that comes with your Mac. Turn off all the autocorrect settings that transform straight quotes into curly quotes. You only want straight quotes for this. (A script editor does this for you automatically.)



Always type at the top of each script something called a shebang: called hash + bang #!. Followed by /bin/zsh (or bash or whichever script type you are using), so that when you use your scripts, Terminal will know to use seashell (or whichever script type you are using) as an interpreter.

That's not very exciting, so let's have a little fun.

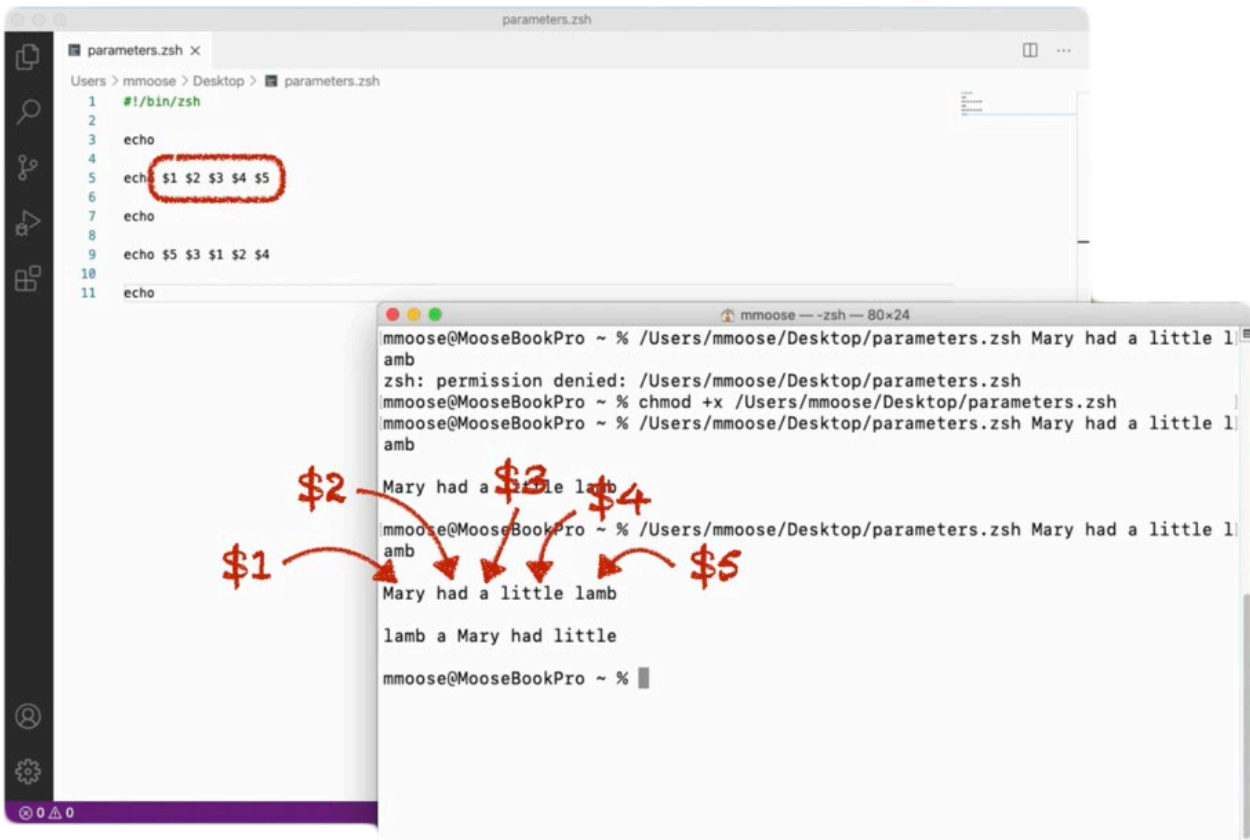
Back in the script, add another echo command. But this time, mix up the order of those script variables:

```
#!/bin/zsh
echo
echo $1 $2 $3 $4 $5
echo
echo $5 $3 $1 $2 $4
echo
```

Now save, and run the script again in Terminal followed by "Mary had a little lamb."

```
~/Desktop/parameters.zsh Mary had a little lamb
```

Here's what you'll see:



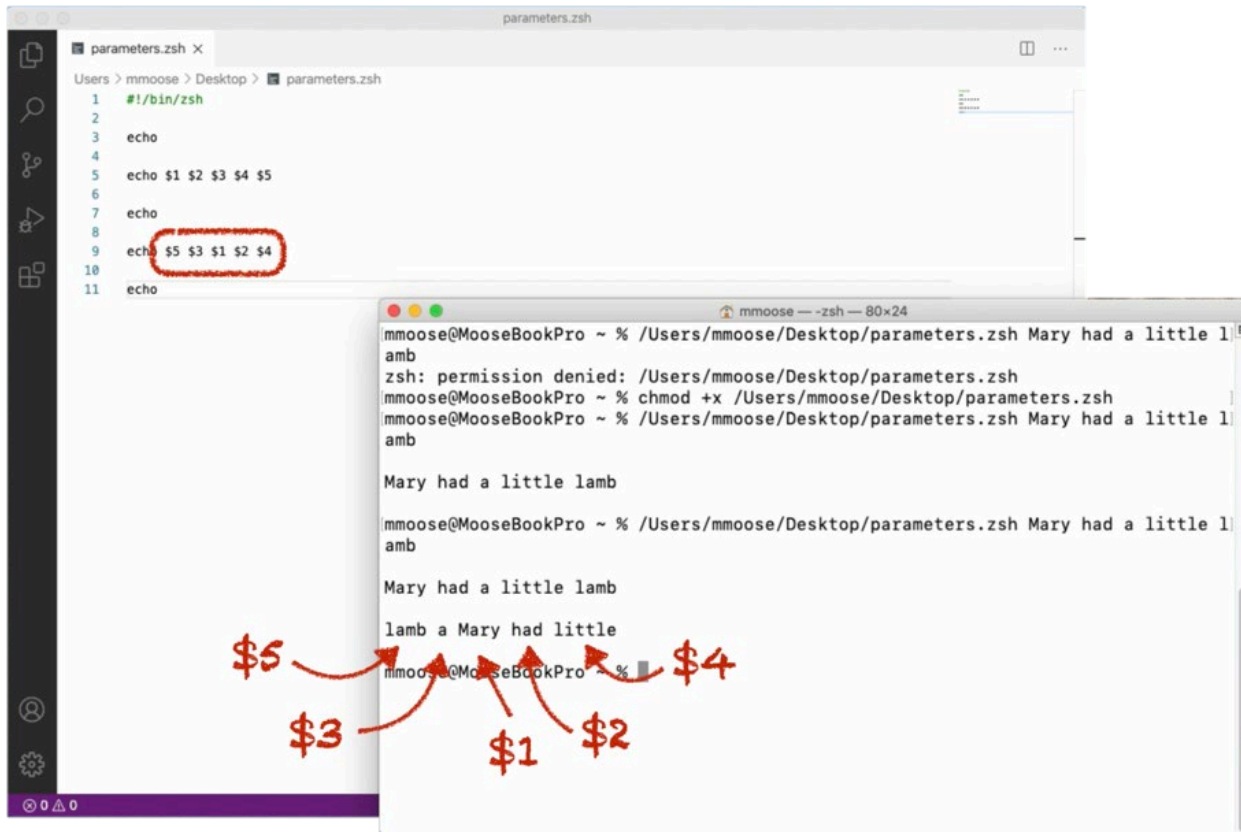
The first line looks right but now the second one is all mixed up. What that's telling us is that each of those number variables in the script — \$1-2-3-4-5 — corresponds to the order of items that we add at the end of the script.

"Mary" is the first word and is "\$1."

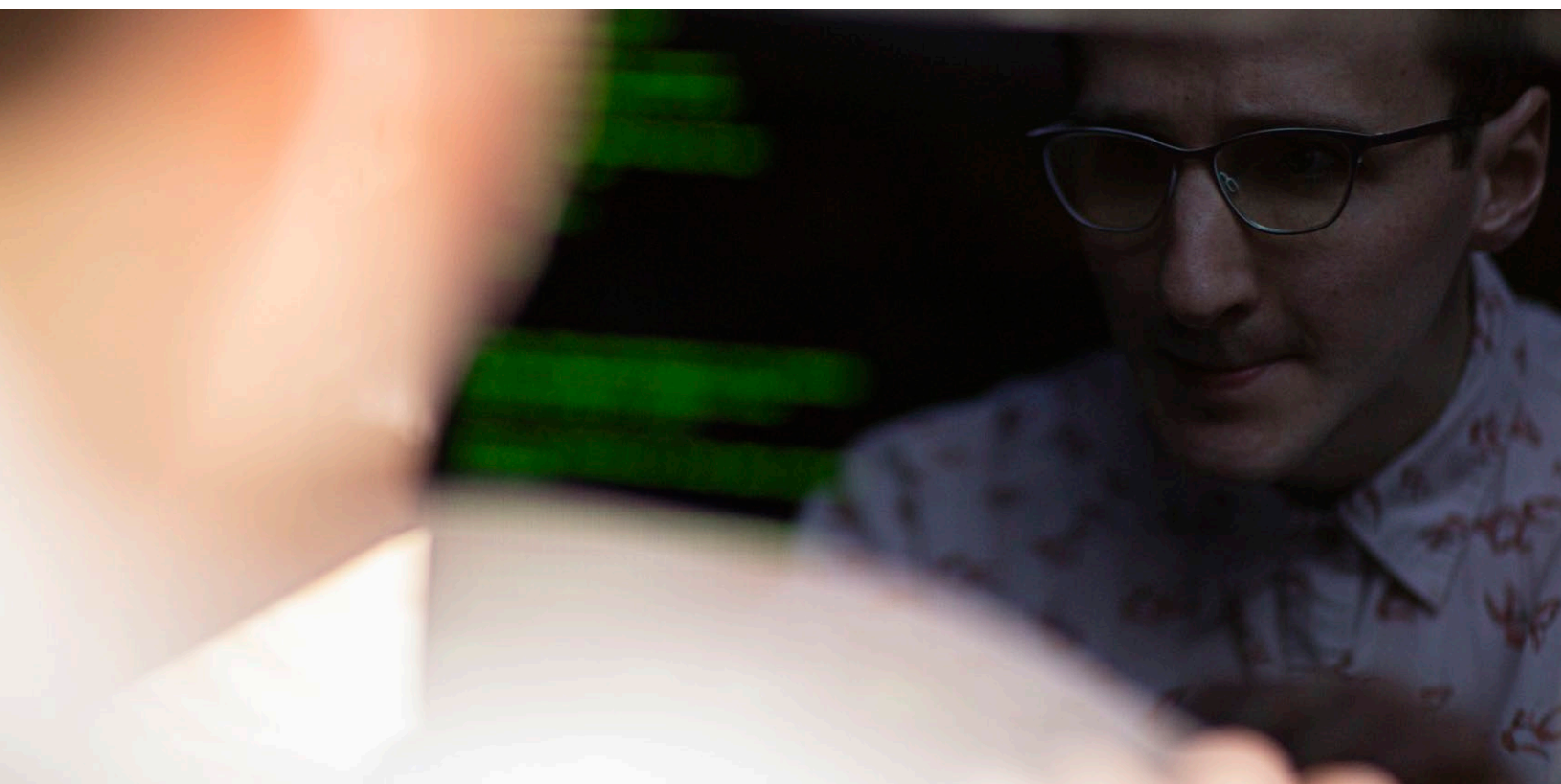
"had" is the second word and is "\$2." etc.

Each of the words in "Mary had a little lamb" is a "script parameter." And we can reference each script parameter by its position following the script.

If you change the order of the variables in the script, then the order of the words changes when you run the script.



Now, that we know that script parameters are about the order of items that follow a script, how do we use this in Jamf Pro? Let's take a look.





## Finding existing scripts

You can find all sorts of existing scripts that other Jamf Pro administrators have created at [jamfnation.com: resources → Jamf Pro add-ons → Scripts](https://jamfnation.com/resources/Jamf-Pro-add-ons/Scripts).

From here, you may browse, or use the search function for whatever it is you'd like to do.

In this example case, we will be using a "time zone" parameter script. Here's what you'll do:

- Download script
- Open file
- Select entire contents
- Copy

In your Jamf Pro instance, navigate to:

### Settings > Computer Management > Scripts > New Script

- Name the new script "Set time zone."
- Under the Script tab, paste the script.
- Under the Options tab, click into the "Parameter 4" field and set the label name to something like "Time Zone." You can name it anything that you want, but it makes sense to name it something meaningful. In this case, when you run the script, you're going to set the time zone that you want in the fourth parameter or "\$4."
- Save.
- Select the Script tab again.
- Find the command for listing time zones that you want to copy.



So, why do these start with parameter "4" and not "1?"

If you look very closely at the tiny type just above the parameter labels, you'll see some text that says "Parameters 1 through 3 are predefined as mount point, computer name and username."

That means Jamf Pro will always send that information as the first three parameters with every script, whether the script uses them or not.

You don't have control over those parameters, but you do have control over the rest. You have up to 8 more parameters that you can define any way you want.

The screenshot shows the Jamf Pro interface for creating a new script. On the left, a sidebar displays system statistics: VERSION 10.22.0-t1591219900, MANAGED Computers: 5, Mobile Devices: 6, UNMANAGED Computers: 0, Mobile Devices: 6. The main content area is titled 'Script Contents' and has tabs for General, Script, Options, and Limitations. The 'Script Contents' tab is active, showing a code editor with the following script content:

```
49 #
50 # SYNOPSIS
51 # sudo setTimeZone.sh
52 # sudo setTimeZone.sh <mountPoint> <computerName> <currentUser> <timeZone>
53 #
54 # If the $timeZone parameter is specified (parameter 4), this is the time zone that will be set.
55 #
56 # If no parameter is specified for parameter 4, the hardcoded value in the script will be used.
57 #
58 # DESCRIPTION
59 # This script sets the system time zone as reflected in the Date & Time preference pane with the
60 # System Preferences application. It has been designed to work on Mac OS X 10.3 and higher.
61 #
62 # A list of supported time zone entries can be found by running the command:
63 #
64 # For Mac OS X 10.5 and later:
65 #
66 # /usr/sbin/systemsetup -listtimezones
67 #
68 # For Mac OS X 10.4 or earlier:
69 #
70 # /System/Library/CoreServices/RemoteManagement/ARDAgent.app/Contents/Support/systemsetup -listtimezones
71 #
72 # The system time zone will be set according to the value specified in the parameter $timeZone.
```

The command `/usr/sbin/systemsetup -listtimezones` is circled in red in the original image.

## How to run the command

- Open Terminal
- Enter “sudo,” which means run with elevated privileges, and then paste in the command:

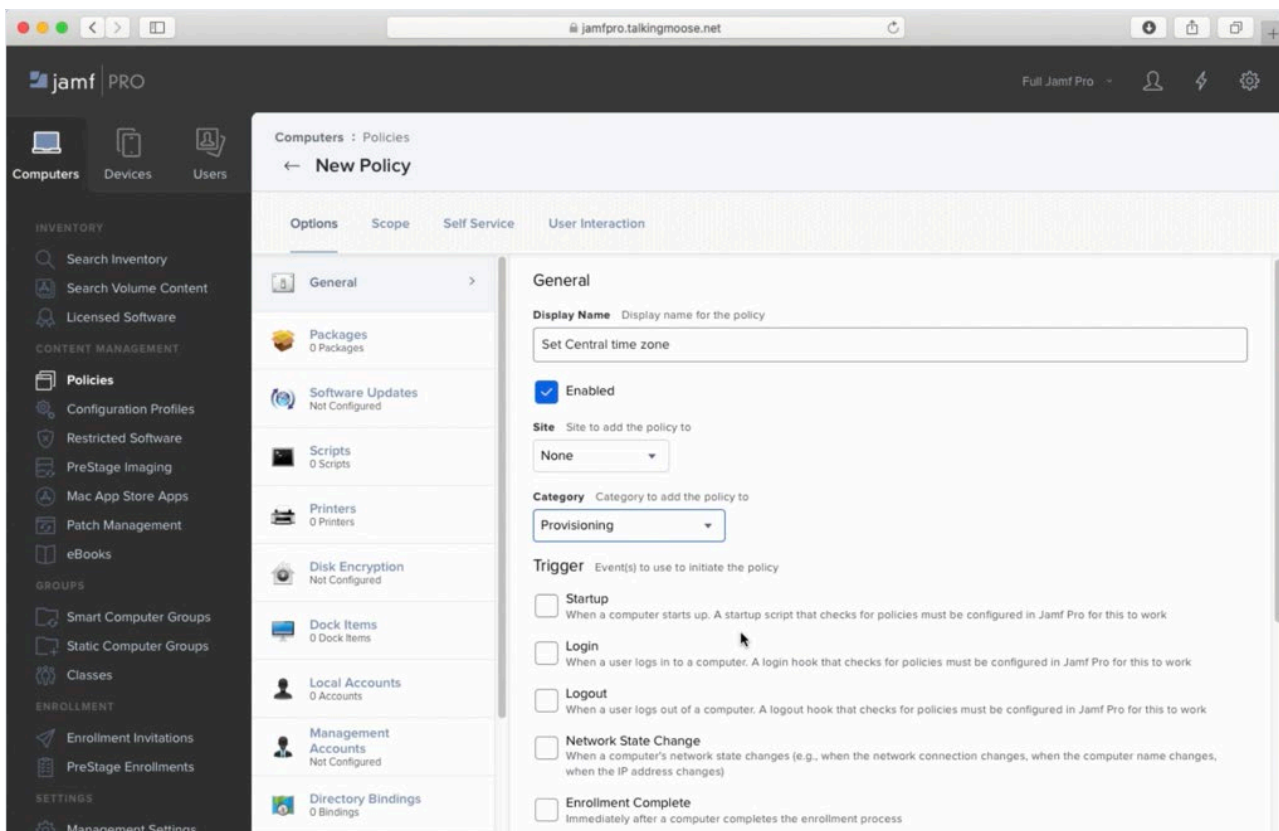
```
/usr/sbin/systemsetup -listtimezones
```

This will show a long list of available time zones in the correct format for the script. In this example, we choose Chicago, which is representative of the Central time zone in the United States, and copy it.

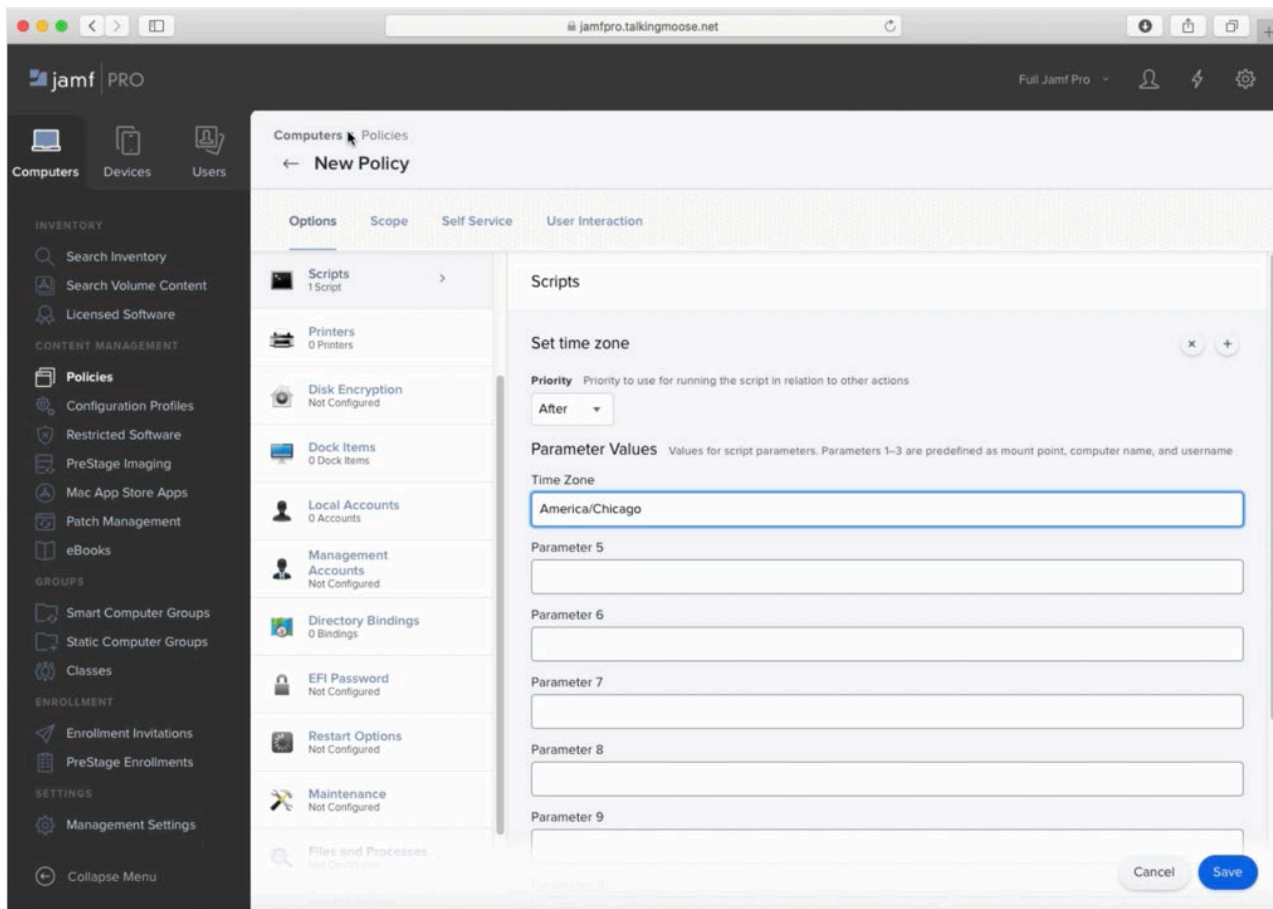
## Creating a policy

Now, let’s go create a policy to run our script.

- In your Jamf instance, Select “Policies” from the left-hand menu, and then the category. In this example, we’ll add it to the “Provisioning” category.
- Name it “Set Central Time Zone,” and add it to the “Provisioning” category.



- Set a custom trigger to let me call this policy by name in a script later, such as setcentraltimezone.
- Select the Scripts option and paste in the time zone that you copied from the Terminal command.



Notice this field is showing the label we added under the Options tab when we pasted in the script.

The label is telling you what you should put here.

All you need to do now is scope the policy and save it.

And now you have a new policy in your “Provisioning” category.

Script parameters let us do something really cool.

Because the script was written to accept script parameters, you can reuse it as many times as you like for different time zones. All you have to do is create a new policy for each time zone, add the same script and then fill out the Time Zone value.

# Create dialogs with jamfHelper and osascript

Dialogs are useful not only for displaying a message to your end users, but also for requesting information from them.

Here are two ways to make dialogs, each with its own advantages.

## jamfHelper

jamfHelper is a command line tool.

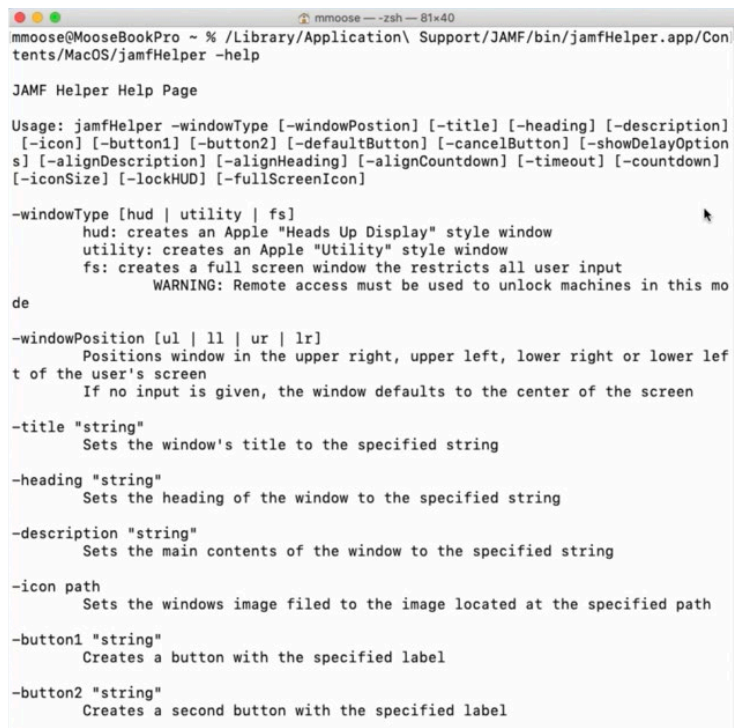
Open Terminal. Then:

- Navigate to Library > Application Support > JAMF > bin > jamfHelper
- Right-click jamfHelper and choose 'Show Package Contents.'
- Navigate to Contents > MacOS where you'll find the command line tool.
- Drag it into Terminal.

At the end of the command line that populates, add "-help" and press return:

```
/Library/Application\ Support/JAMF/bin/jamfHelper.app/  
Contents/MacOS/jamfHelper -help
```

This will show you everything you need to know to use jamfHelper.



```
m MooseBookPro -- zsh -- 81x40  
m MooseBookPro ~ % /Library/Application\ Support/JAMF/bin/jamfHelper.app/Con  
tents/MacOS/jamfHelper -help  
  
JAMF Helper Help Page  
  
Usage: jamfHelper [-windowType] [-windowPosition] [-title] [-heading] [-description]  
[-icon] [-button1] [-button2] [-defaultButton] [-cancelButton] [-showDelayOption  
s] [-alignDescription] [-alignHeading] [-alignCountdown] [-timeout] [-countdown]  
[-iconSize] [-lockHUD] [-fullScreenIcon]  
  
-windowType [hud | utility | fs]  
    hud: creates an Apple "Heads Up Display" style window  
    utility: creates an Apple "Utility" style window  
    fs: creates a full screen window the restricts all user input  
    WARNING: Remote access must be used to unlock machines in this mo  
de  
  
-windowPosition [ul | ll | ur | lr]  
    Positions window in the upper right, upper left, lower right or lower lef  
t of the user's screen  
    If no input is given, the window defaults to the center of the screen  
  
-title "string"  
    Sets the window's title to the specified string  
  
-heading "string"  
    Sets the heading of the window to the specified string  
  
-description "string"  
    Sets the main contents of the window to the specified string  
  
-icon path  
    Sets the windows image filed to the image located at the specified path  
  
-button1 "string"  
    Creates a button with the specified label  
  
-button2 "string"  
    Creates a second button with the specified label
```

Because the path to jamfHelper is so long, you can put it into a shorter variable name called just jamfHelper with this script:

```
#!/bin/zsh
```

```
jamfHelper="/Library/Application  
Support/JAMF/bin/jamfHelper.app/Contents/MacOS/  
jamfHelper"
```

From here on, to call jamfHelper, you just need to put a dollar sign in front of your variable name and wrap it in double-quotes:

```
"$jamfHelper"
```

## What can jamfHelper do?

Let's say you want to add some options to jamfHelper. The first one is to define a window type — it supports three. To start, we'll go with "Heads-Up Display." Add `-windowType` to your script, and the type of window: "hud."

```
"$jamfHelper" -windowType hud \
```

Next, add a heading. This must be in quotes:

```
-heading "Preparing your computer..." \
```

Then, add a description.

```
-description "installing Microsoft Office 2019"\
```

If you want to add interest to the dialog box, add an icon. Choose an icon and put its path here. Put the path in quotes in case it contains spaces.

```
-icon "/System/Library/CoreServices/Finder.app/  
Contents/Resources/Finder.icns"
```



Here's a nice trick to get the correct pathname. Right-click jamfHelper and point to "Copy," but then also press the Option key to copy its path instead. In this script, all you need to do is paste the path between the quotes.

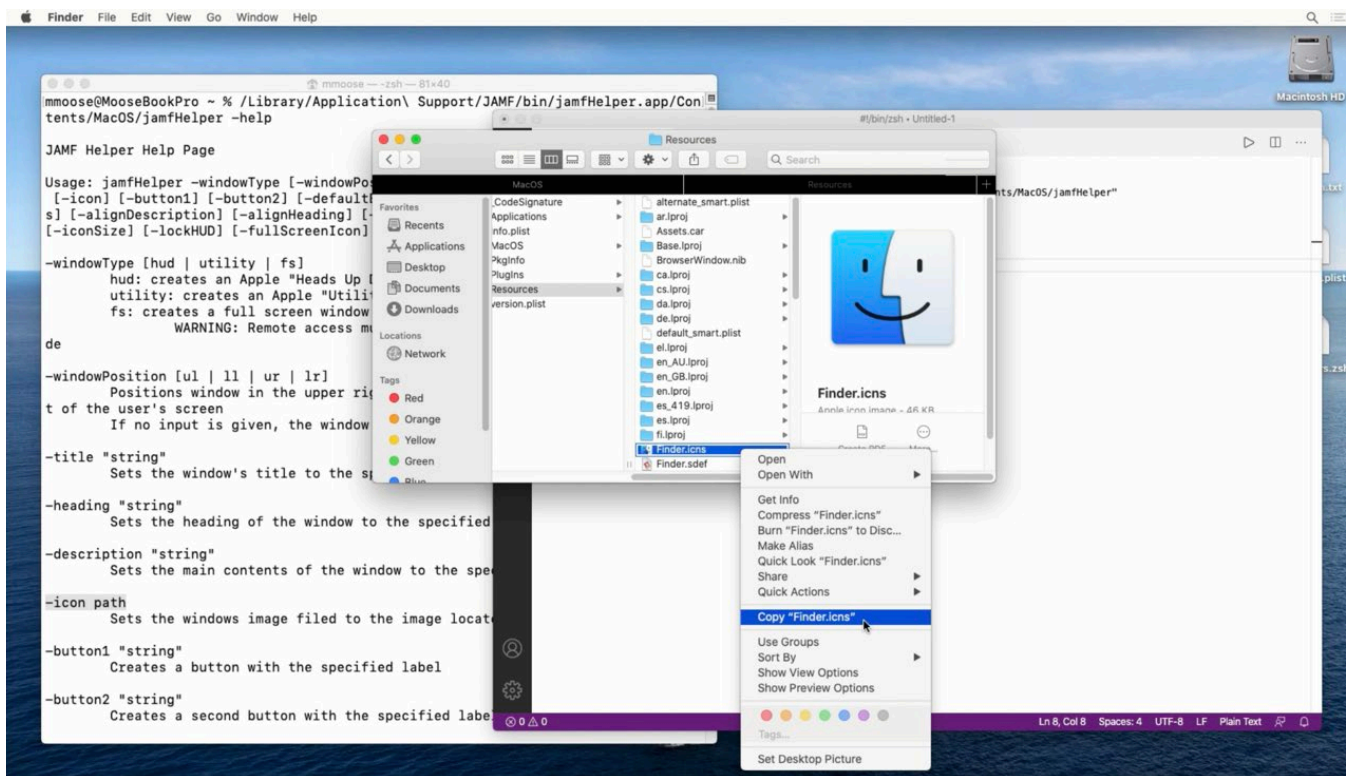


A backslash at the end of a line allows you to wrap what would normally be a very long command onto multiple lines.



## How to find already existing icons on your Mac

- In the Finder, navigate to System > Library > CoreServices and locate the Finder app.
- Right-click the Finder, choose “Show Package Contents” and then Contents > Resources
- Select the Finder icon.
- Right-click the Finder icon, hold the Option key and copy the path.
- Then, paste the path into your script.



Here is how it all looks together:

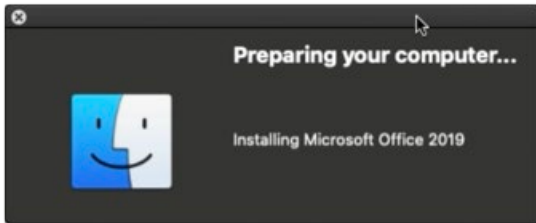
```
#!/bin/zsh

jamfHelper="/Library/Application
Support/JAMF/bin/jamfHelper.app/Contents/MacOS/jamfHelper"

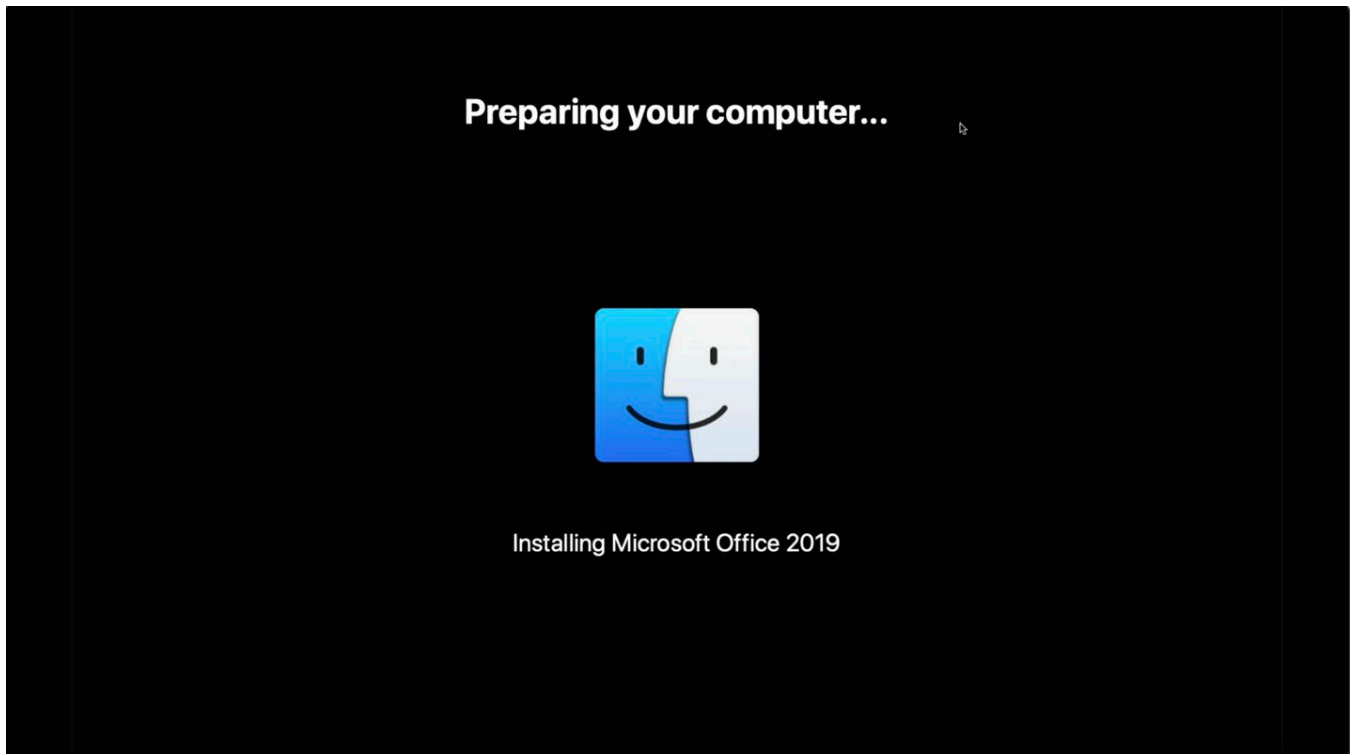
"$jamfHelper" -windowType hud \
-heading "Preparing your computer..." \
-description "installing Microsoft Office 2019"\
-icon "/System/Library/CoreServices/Finder.app/Contents/Resources/Finder.icns"
```

Run your script directly from your script editor.

You'll get a very simple dialog:



For a full-screen dialog, change the window type from “hud” to “fs”, which stands for “Full Screen,” and run it again:



A full screen dialog like this is a great way to prevent the person in front of the computer from using it while it's provisioning. It can also update to inform the user of progress as tasks like installing software run behind it.

If at any time you need to see what's happening behind the dialog, just press Command + q to quit.

## The osascript Command

You can also create dialogs through AppleScript with the `osascript` command from the Terminal.

It can do something that `jamfHelper` can't.

First, create a command for a “choose” dialog, including three department names.

You must use double-quotes to begin and end this command, and when you have double-quotes as part of the command as well, you'll need to add backslashes in front of them to treat them as literal double-quotes — not the double-quotes surrounding the command:

```
asCommand="choose from list {\\"Accounting\\", \\"Sales\\", \\"kiosk\\"}
```

Then, add a message to display to your end user, explaining what the Mac is about to do:

```
with prompt \\"Hello, let's prepare your Mac.  
To start, choose your department below...\\\"
```

The last part of the command adds a title to the dialog window. Remember to put your closing double-quote at the end of the command:

```
with title \\"Prepare your Mac\\\""
```

This line actually tells `osascript` to run the command and then puts the result into the “department” variable:

```
department=$( /usr/bin/osascript -e \"$asCommand\" )
```

Last, echo the “department” variable to show what the user selected:

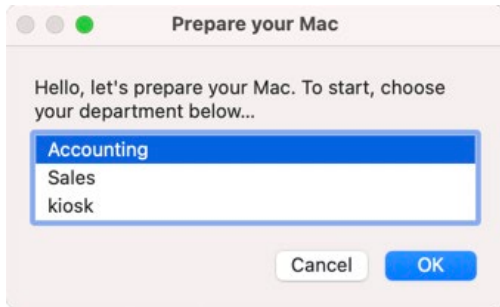
```
echo \"$department\"
```

Here's what the whole script looks like:

```
#!/bin/zsh  
  
asCommand="choose from list {\\"Accounting\\", \\"Sales\\", \\"kiosk\\"} with prompt \  
\"Hello, let's prepare your Mac. To start, choose your department below...\\\"  
with title \\"Prepare your Mac\\\""  
  
department=$( /usr/bin/osascript -e \"$asCommand\" )  
  
echo \"$department\"
```



Now, test the script in your script editor. Here the user chose 'Accounting,' and this is the result they will see:



AppleScript and osascript can do a lot more with dialogs, but this is a good way to start.

## Putting it all together

Here's a script you can add to Self Service as a method to let an onboarding new employee provision their Mac by choosing his or her department.

First, it collects the logged-in user's full name and username. It will then use the full name in the osascript prompt to choose a department:

```
jamfHelper="/Library/Application Support/JAMF/bin/jamfHelper.app/Contents/MacOS/jamfHelper"

currentUser=$( /usr/bin/stat -f "%Su" /dev/console )

fullName=$( /usr/bin/id -F "$currentUser" ) # e.g. "mmoose"

echo "Provisioning user is $fullName ($currentUser)"
```

Then, it asks the user to choose a department:

```
asCommand="choose from list {\\"Accounting\\", \\"Sales\\", \\"Kiosk\\"} with prompt \\"Hello, $fullName! Let's prepare your Mac. To start, choose your department below...\\" with title \\"Prepare your Mac\\""

department=$( /usr/bin/osascript -e "$asCommand" )

echo "Provisioned for department $department"
```

Another osascript asks for the asset tag of the computer:

```
asCommand="text returned of (display dialog \"Enter this Mac's asset tag (see
bottom of computer)...\" default answer \"\" with title \"Prepare your Mac\")"

assetTag=$( /usr/bin/osascript -e "$asCommand" )

echo "Device asset tag is $assetTag"
```

The script will then set the time zone for all computers:

```
/usr/local/bin/jamf policy -event settimezonechicago
```

And then it'll evaluate the chosen department and install the necessary software:

```
if [[ "$department" = "Accounting" ]]; then
    echo "Provisioning this Mac for Accounting"

    /usr/local/bin/jamf policy -event installchrome
    /usr/local/bin/jamf policy -event installoffice

elif [[ "$department" = "Sales" ]]; then

    echo "Provisioning this Mac for Sales"

    /usr/local/bin/jamf policy -event installchrome
    /usr/local/bin/jamf policy -event installoffice
    /usr/local/bin/jamf policy -event installzoom

else

    echo "Provisioning this Mac as a kiosk"

    /usr/local/bin/jamf policy -event installchrome
    /usr/local/bin/jamf policy -event installzoom

fi
```

Once the software is installed, it'll run an inventory update that'll include the asset tag, department and username of the computer:

```
"$jamfHelper" -windowType "fs" \  
-heading "Preparing your Mac..." \  
-description "Updating inventory" \  
-icon  
"/System/Library/CoreServices/Finder.app/Contents/Resources/Finder.icns" &  
  
/usr/local/bin/jamf recon -assetTag "$assetTag" -department "$department"  
-endUsername "$currentUser"
```

It will then create a folder in Library for administrative tools and information:

```
"$jamfHelper" -windowType "fs" \  
-heading "Preparing your Mac..." \  
-description "Creating admin folder" \  
-icon  
"/System/Library/CoreServices/Finder.app/Contents/Resources/Finder.icns" &  
  
/bin/mkdir -p "/Library/Talking Moose Industries"
```

And leave a provisioning receipt behind:

```
"$jamfHelper" -windowType "fs" \  
-heading "Preparing your Mac..." \  
-description "Writing provisioning receipt" \  
-icon  
"/System/Library/CoreServices/Finder.app/Contents/Resources/Finder.icns" &  
  
/usr/bin/defaults write "/Library/Talking Moose Industries/receipt.plist"  
provisiondate -date $( /bin/date "+%Y-%m-%d" )  
/usr/bin/defaults write "/Library/Talking Moose Industries/receipt.plist"  
provisioner -string "$currentUser"  
/usr/bin/defaults write "/Library/Talking Moose Industries/receipt.plist"  
department -string "$department"  
/usr/bin/defaults write "/Library/Talking Moose Industries/receipt.plist"  
assettag -string "$assetTag"
```

Finally, it'll inform the user and then reboot the computer:

```
"$jamfHelper" -windowType "fs" \  
-heading "Preparing your Mac..." \  
-description "Restarting your Mac in one minute" \  
-icon  
"/System/Library/CoreServices/Finder.app/Contents/Resources/Finder.icns" &  
/sbin/shutdown -r +1 &  
  
exit 0
```

Full script, with descriptors embedded:

```
#!/bin/zsh  
  
# assign long path to JamfHelper to a shorter variable  
  
jamfHelper="/Library/Application  
Support/JAMF/bin/jamfHelper.app/Contents/MacOS/jamfHelper"  
  
# get information about the current user  
  
currentUser=$( /usr/bin/stat -f "%Su" /dev/console )  
fullName=$( /usr/bin/id -F "$currentUser" ) # e.g. "Martin Moose"  
  
echo "Provisioning user is $fullName ($currentUser)"  
  
# AppleScript command to ask the currently logged in user to choose a department  
  
asCommand="choose from list {\"Accounting\", \"Sales\", \"Kiosk\"} with prompt  
\"Hello, $fullName! Let's prepare your Mac. To start, choose your department  
below...\" with title \"Prepare your Mac\""  
  
# run the command  
  
department=$( /usr/bin/osascript -e "$asCommand" )  
  
echo "Provisioned for department $department"  
  
# AppleScript command to ask the currently logged in user to enter an asset tag  
  
asCommand="text returned of (display dialog \"Enter this Mac's asset tag (see  
bottom of computer)...\" default answer \"\" with title \"Prepare your Mac\")"  
  
# run the command  
  
assetTag=$( /usr/bin/osascript -e "$asCommand" )  
  
echo "Device asset tag is $assetTag"
```

```

# build this Mac for the selected department

# global settings

/usr/local/bin/jamf policy -event settimezonechicago

if [[ "$department" = "Accounting" ]]; then

    echo "Provisioning this Mac for Accounting"

    /usr/local/bin/jamf policy -event installchrome
    /usr/local/bin/jamf policy -event installoffice

elif [[ "$department" = "Sales" ]]; then

    echo "Provisioning this Mac for Sales"

    /usr/local/bin/jamf policy -event installchrome
    /usr/local/bin/jamf policy -event installoffice
    /usr/local/bin/jamf policy -event installzoom

else

    echo "Provisioning this Mac as a kiosk"

    /usr/local/bin/jamf policy -event installchrome
    /usr/local/bin/jamf policy -event installzoom

fi

# update Jamf Pro inventory

"$jamfHelper" -windowType "fs" \
-heading "Preparing your Mac..." \
-description "Updating inventory" \
-icon
"/System/Library/CoreServices/Finder.app/Contents/Resources/Finder.icns" &

/usr/local/bin/jamf recon -assetTag "$assetTag" -department "$department" -
endUsername "$currentUser"

# create admin folder

"$jamfHelper" -windowType "fs" \
-heading "Preparing your Mac..." \
-description "Creating admin folder" \
-icon
"/System/Library/CoreServices/Finder.app/Contents/Resources/Finder.icns" &

/bin/mkdir -p "/Library/Talking Moose Industries"

```

```
# create provisioning receipt

"$jamfHelper" -windowType "fs" \
-heading "Preparing your Mac..." \
-description "Writing provisioning receipt" \
-icon
"/System/Library/CoreServices/Finder.app/Contents/Resources/Finder.icns" &

/usr/bin/defaults write "/Library/Talking Moose Industries/receipt.plist"
provisiondate -date $( /bin/date "+%Y-%m-%d" )
/usr/bin/defaults write "/Library/Talking Moose Industries/receipt.plist"
provisioner -string "$currentUser"
/usr/bin/defaults write "/Library/Talking Moose Industries/receipt.plist"
department -string "$department"
/usr/bin/defaults write "/Library/Talking Moose Industries/receipt.plist"
assettag -string "$assetTag"

"$jamfHelper" -windowType "fs" \
-heading "Preparing your Mac..." \
-description "Restarting your Mac in one minute" \
-icon
"/System/Library/CoreServices/Finder.app/Contents/Resources/Finder.icns" &

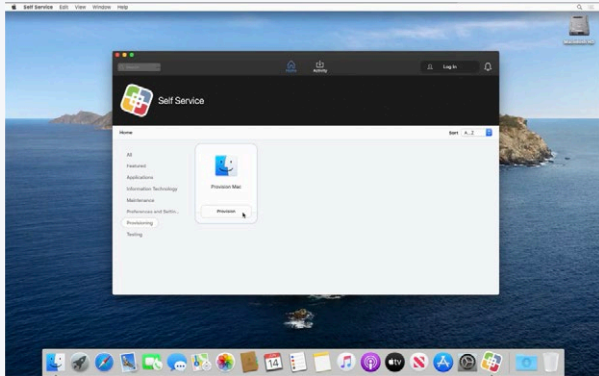
# restart the Mac
/sbin/shutdown -r +1 &

exit 0
```



Let's see what it all looks like!

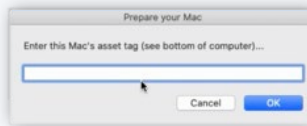
With Self Service open, start the process by clicking the Provision button.



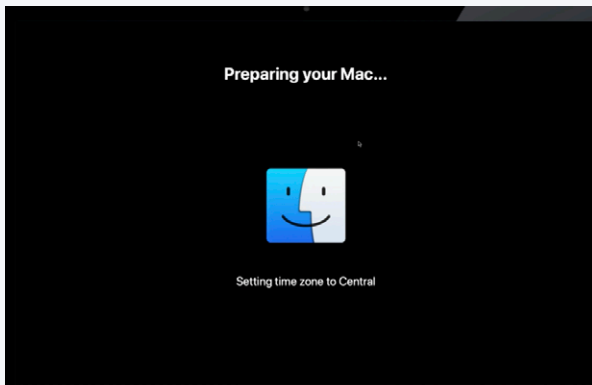
STEP 1.



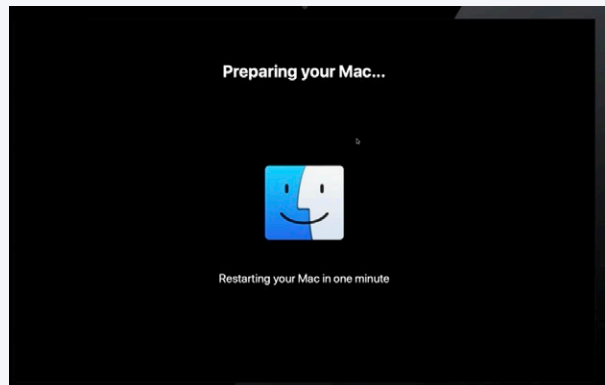
STEP 2.



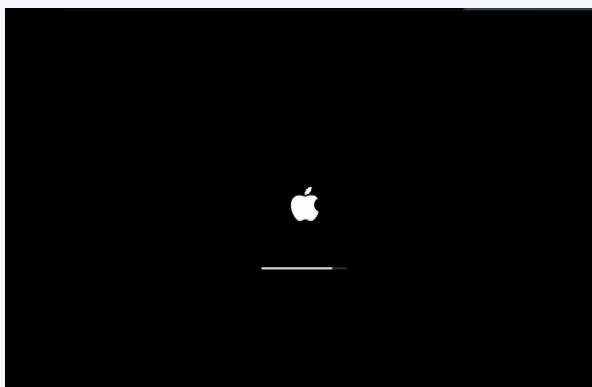
STEP 3.



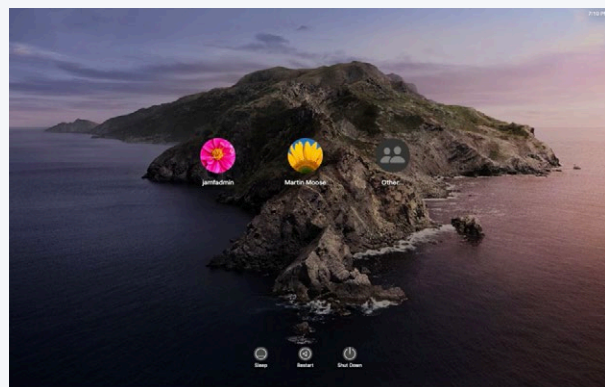
STEP 4.



STEP 5.



STEP 6.

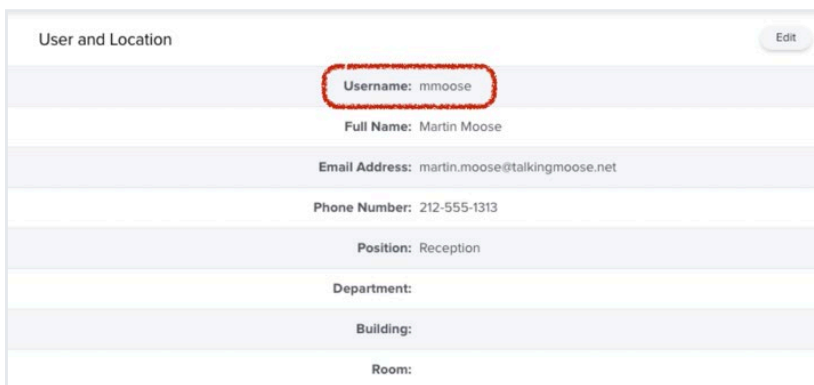
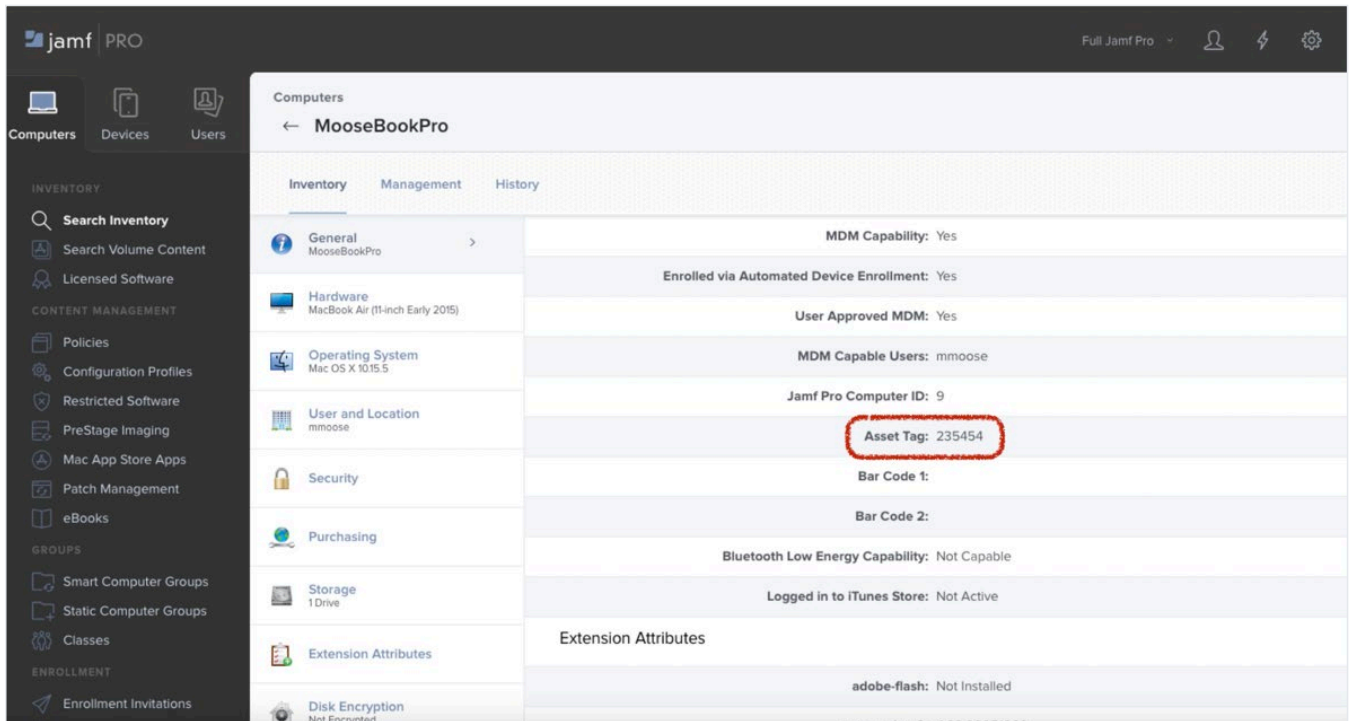


STEP 7.

When provisioning is done, the computer reboots and the Mac is back at the login window, ready to go.

After the user logs in, their software is installed and they can see their provisioning receipt in the Library folder.

When the Mac admin views the computer record in Jamf Pro, the asset tag is populated from the osascript dialog, the username field is populated — which allows the admin to do an LDAP lookup and pull in additional information such as the email address, phone number, user principal name and other details from Active Directory.



And that's it!



## Further resources and ways to learn

### Jamf's Training Catalog

As you keep scripting, don't forget, all Jamf customers have access to our entire training catalog with more than 15 hours of short how-to videos that cover everything from help desk to engineer to administrator: [trainingcatalog.jamf.com](https://trainingcatalog.jamf.com)

Check out the Scripting Series with 15 modules and video lessons that each take less than 30 minutes to complete. This will help you keep learning how to script.

### Github

Explore the open-source community on [GitHub.com](https://github.com). Open-source software is free to use, and you'll find hundreds of scripts and projects on Jamf's GitHub page: [github.com/jamf](https://github.com/jamf)

### Resources from the author Bill Smith

If you want to see a lot of examples of short and long scripts for a variety of needs, check out Bill Smith's [gist repository on GitHub](https://gist.github.com/talkingmoose). This is where you'll find the sample provisioning script from this white paper and some of the other code he used here as well: <https://gist.github.com/talkingmoose>

We hope this guide has helped you to push your scripting skills to the next level.

Scripting combined with a good Enterprise Management System can really make the difference between hours and hours of work and the click of a button.

[Request Trial](#)

We'd love to suggest Jamf Pro.

