

Table Of Contents

Section 1: Introduction

In section 1 I'll explain what you're going to need etc

Section 1.1: Installing and setting up theos (macOS)

In section 1.1 I'll show you how to install and set up theos which we'll need to make Tweaks

Section 2: Setting Up Our Project

In section 2 I'll show you how to create your first project with theos

Section 3: Installing Our IDE

In section 3 I'll give you some advice about an IDE to use for the coding part

Section 4: Modifying Our Files

In section 4 I'll show you what each file does and what needs to be changed
Section: 4.1: Modifying Our Control File
Section 4.2: Modifying The Makefile

Section 5: Creating Our First Tweak

In Section 5 I'll teach you how to create your first little tweak by also explaining every step

Section 6: Layout Subviews

In section 6 I'll talk about layoutSubviews and how you could use them in a tweak

Section 7: Extension For Our Haptic Volume Tweak

In section 7 we're going to add more options to our haptics tweak as well as preferences (cephei)

Side Section: Creating Your Own Repo To Host Tweaks And Apps

In the side section I'll show you how to create your own repo (two ways) where you can upload tweaks and apps

Section 1 - Introduction

Tweak development for beginners

If you haven't set up [theos](#) yet, do it before continuing
(section 1.1 shows how to do it)

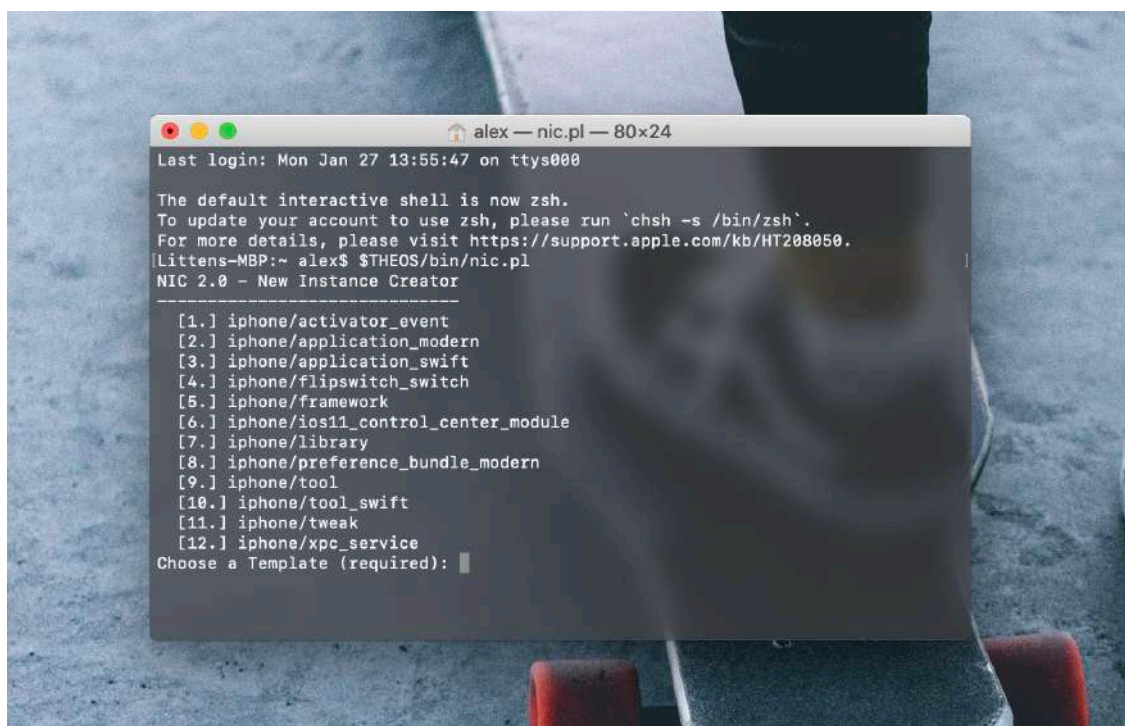
Sooo.. the big question of everyone who wants to develop their own tweaks is “How would I go on doing that?” Or “Where can I learn it?”. The answer is, the internet, there are so many places to learn different kinds of things like Udemy for Video courses, for example Objective-C which is needed to develop tweaks. So now that you know you'll need objective c, you should also keep in mind to look at open source tweaks, to get to know the syntax and way they're built. The things we're going to make are also downloadable here: <https://github.com/Litteen/guideTweaks>

**I AM GOING TO EXPLAIN WHAT YOU'LL NEED THE BEST WAY I CAN, SO LET'S GET INTO IT ^^
YOU CAN ALWAYS ACCESS THIS SITE BUT ALSO FEEL FREE TO DOWNLOAD IT**

We will firstly take a look at how to create a tweak template with the theos environment, so just open your Terminal and type [\\$THEOS/bin/nic.pl](#)

(quick explanation of that command: theos was installed to your home directory and it has also set up an environment variable to access it quickly, the '\$' points to the theos installation directory and after just the way down to the file called 'nic.pl')

You should see this now:



```
alex — nic.pl — 80x24
Last login: Mon Jan 27 13:55:47 on ttys000

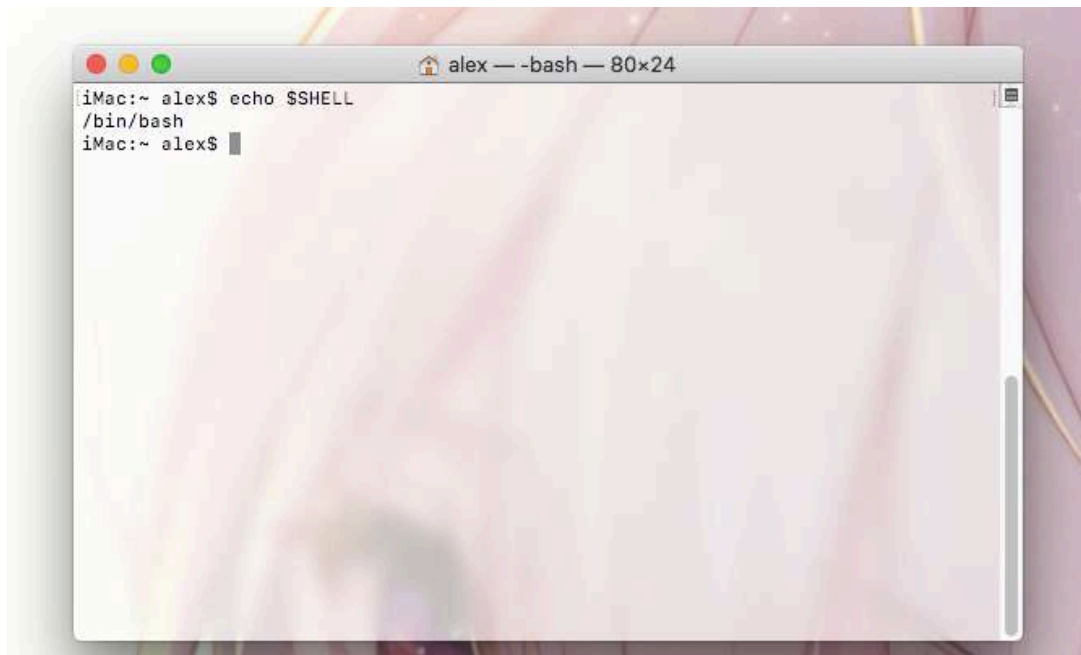
The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
[Littens-MBP:~ alex$ $THEOS/bin/nic.pl
NIC 2.0 - New Instance Creator
-----
[1.] iphone/activator_event
[2.] iphone/application_modern
[3.] iphone/application_swift
[4.] iphone/flipswitch_switch
[5.] iphone/framework
[6.] iphone/ios11_control_center_module
[7.] iphone/library
[8.] iphone/preference_bundle_modern
[9.] iphone/tool
[10.] iphone/tool_swift
[11.] iphone/tweak
[12.] iphone/xpc_service
Choose a Template (required):
```

Section 1.1 - Installing and setting up theos (macOS)

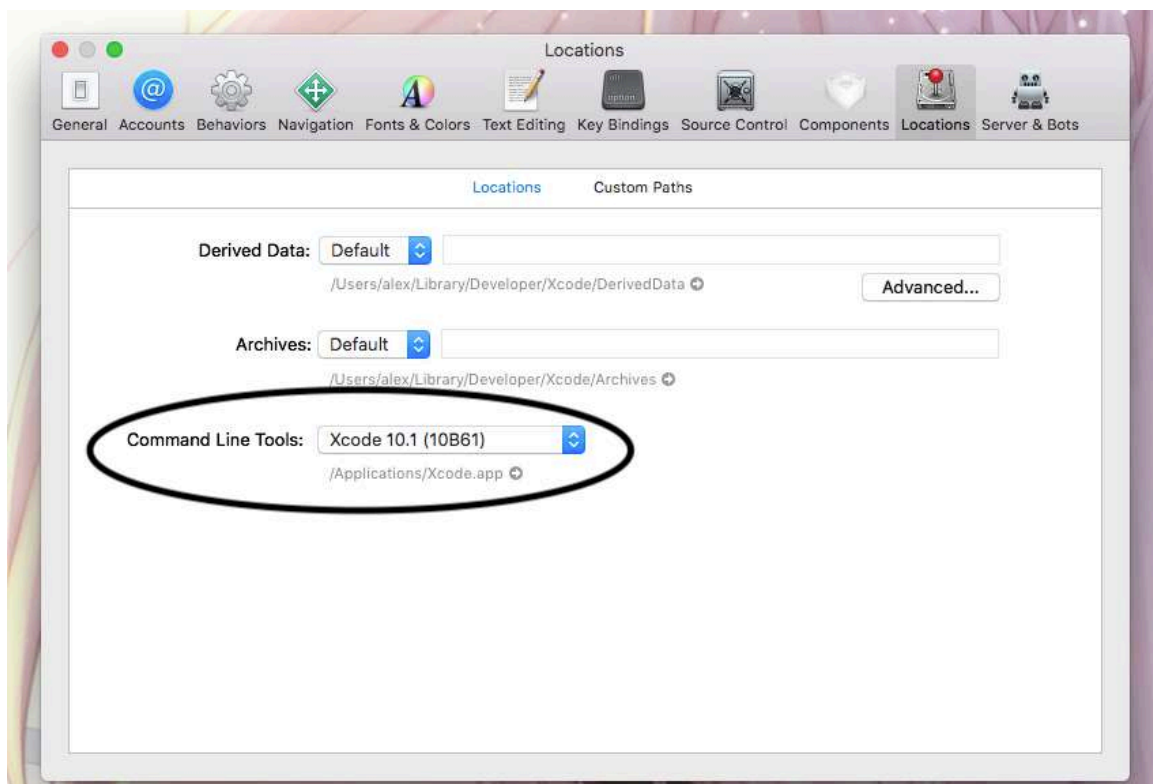
If you want to install it on another platform look it up on the official theos GitHub page, installing on Windows or Linux also means that you won't be able to compile for arm64e (A12) devices

Steps:

- 1) Install Xcode because we'll need the developer tools
- 2) Make sure your terminal profile is bash and not zsh (find out by typing `echo $SHELL` in terminal, and if the current one is zsh type `chsh -s /bin/bash` to change it to bash)



- 3) Make sure your developer tools are properly selected in Xcode



Now that we are ready to install theos let's start by open a new terminal window, and follow these steps:

1) Install homebrew if you haven't installed it yet (homebrew lets you install all different kinds of packages that apple didn't include) by putting in this: `/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"`

2) `brew install ldid xz`

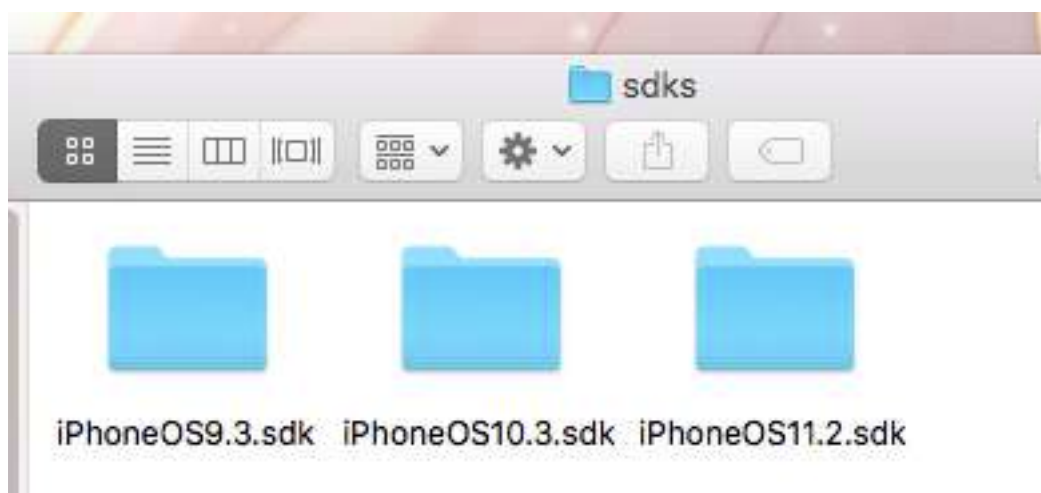
3) `echo "export THEOS=~/.theos" >> ~/.profile`

Now type `echo $THEOS`, if the output is a path continue to step 4, but if nothing appears for the output type `echo "export THEOS=~/.theos" >> ~/.zprofile` (that probably only happen on macOS 10.14 because of the zsh shell)

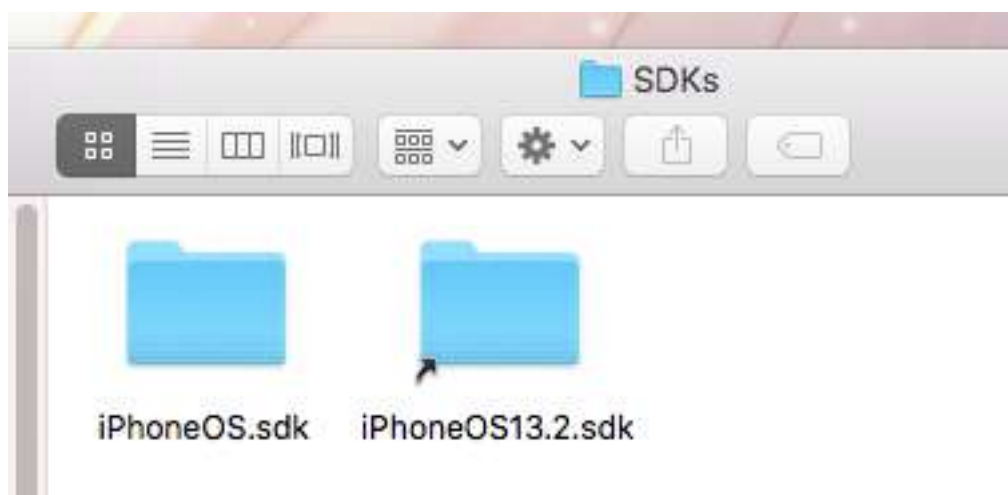
4) `git clone --recursive https://github.com/theos/theos.git $THEOS`

5) For the last step we need to get some sdks to compile, `cd` into your theos directory and then into the sdks directory, then type `git clone https://github.com/theos/sdks.git`

6) type `open sdks/` and drag these three folders to the previous one so that they're in the main sdks folder



To get iOS 13 sdks copy the iPhoneOS.sdk folder and the iPhoneOS13.2 shortcut from Xcode/Contents/Developer/Platforms/iPhoneOS.platform/Developer/SDKs/ to the theos sdks folder



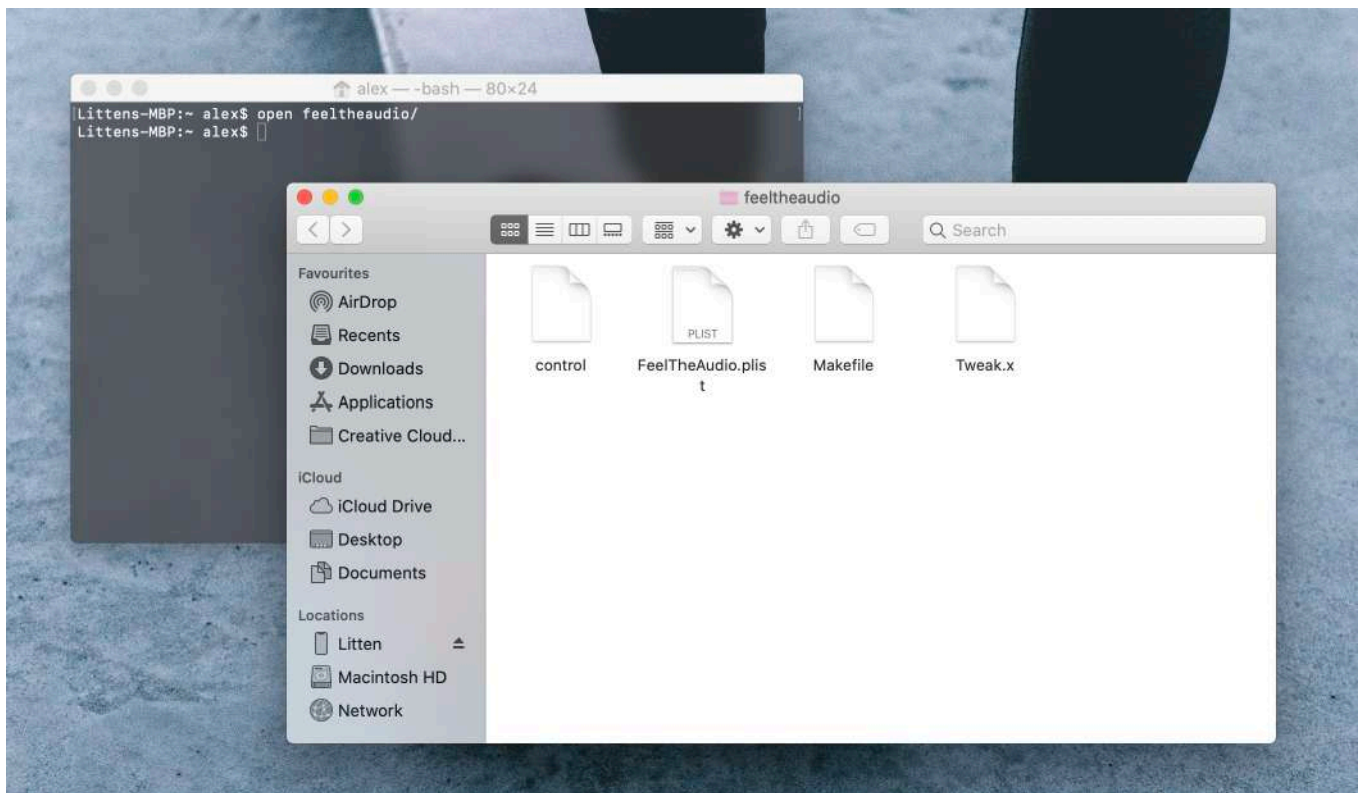
Section 2 - Setting Up Our Project

- At first we'll need to choose what we're going to make, so just put in the number which says "iphone/tweak" next to it (11 for me)
- Secondly it asks us to give the Project (Tweak,..) a name, I'll call this example "FeelTheAudio" (maybe you can guess what we're going to create)
- Thirdly we'll need to give it a bundle identifier, I'll call mine sh.litten.feeltheaudio because of my domain
- Then put in your name
- The last step is to tell theos what the tweak will target, I will write down com.apple.UIKit but you can also leave the default in and just press return on your keyboard
- For the last step just press return

Now we've set up our tweak directory successfully and it's located in your home directory but you can just type `open tweakName` (for me feeltheaudio) in terminal and it will open a finder window located at your tweak

It basically created all files you'll need

- The control file contains information about your package
- The plist contains the target (Bundle) which is com.apple.springboard as default
- The makefile tells theos how and what to compile
- The Tweak.x contains the code for your Tweak



Section 3 - Installing Our IDE

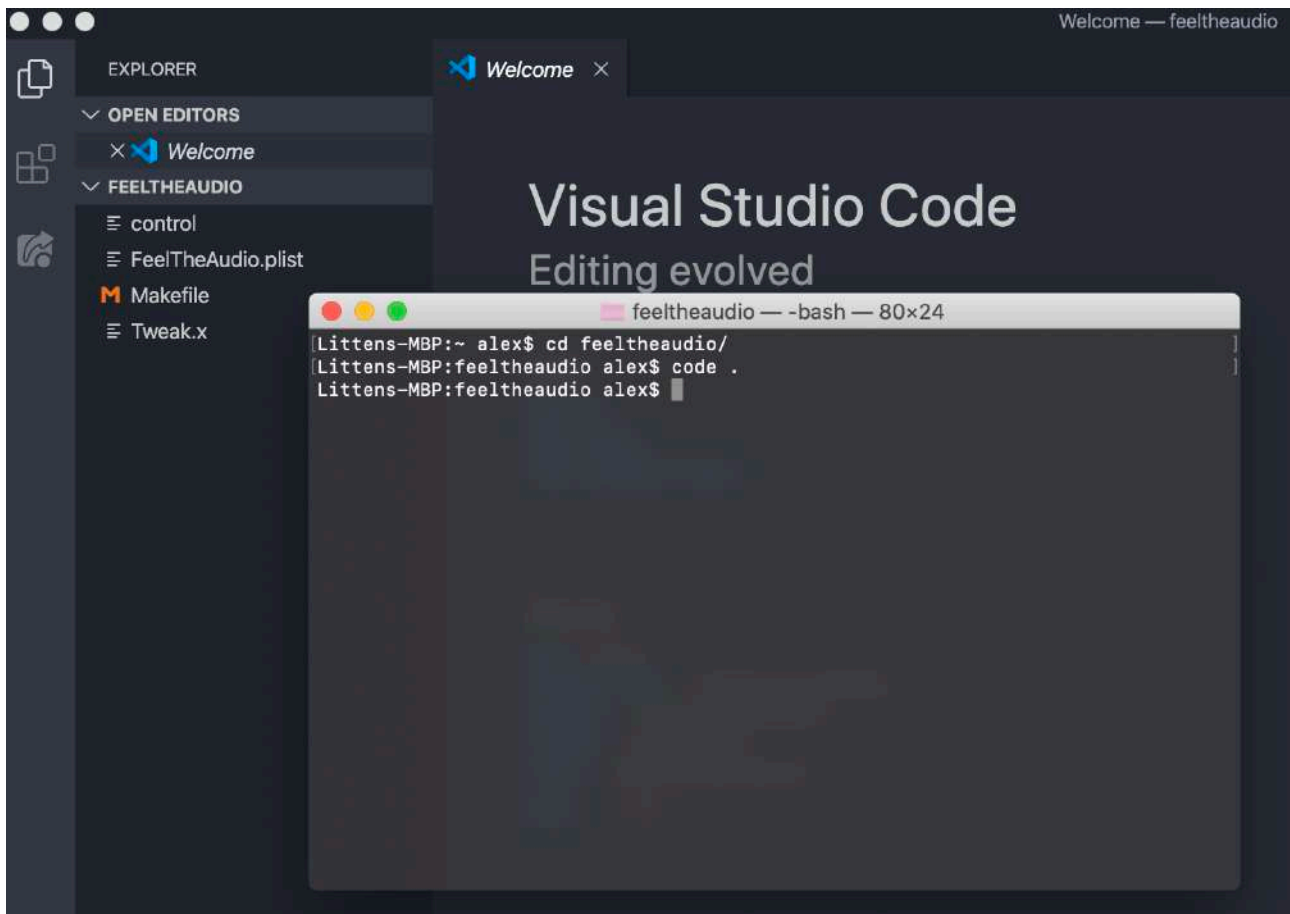
If you prefer to use another IDE just jump to Section 4 but we'll set up VS Code in this section

Go and download Visual Studio Code

Optionally you can install the Terminal extension for it to make it easier to open projects
This will give you the ability to just type `code .` or `code Tweak.x` in terminal to directly open the whole project or file with VS Code from Terminal

To do that open Code and press F1 on your keyboard or Touch Bar, and type `shell command`
Now click on "install code command at PATH" and restart your shell/Terminal to reload the profiles

When you open your terminal again `cd` into your tweak folder and just type `code .` and it will automatically open the Project with Code



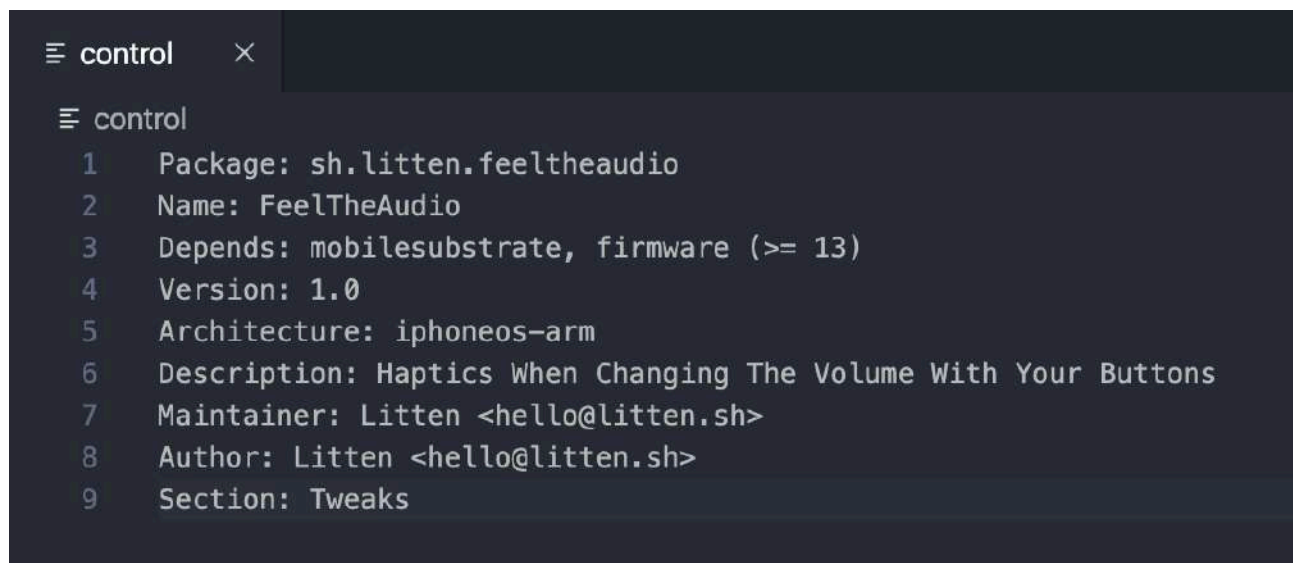
Section 4 - Modifying Our Files

Section 4.1 Modifying Our Control File

Now that we got everything up and running we can start

At first check and if needed modify your control file, I will also put in my email and add a new dependency

Here you will also change the description and other things like conflicts

A screenshot of a code editor with a dark theme. At the top, there is a tab labeled 'control' with a hamburger menu icon on the left and a close 'X' icon on the right. Below the tab, the text 'control' is followed by a list of nine lines of code, each preceded by a number from 1 to 9. The code defines a control file for 'FeelTheAudio' with various metadata and dependencies.

```
1 Package: sh.litten.feeltheaudio
2 Name: FeelTheAudio
3 Depends: mobilessubstrate, firmware (>= 13)
4 Version: 1.0
5 Architecture: iphoneos-arm
6 Description: Haptics When Changing The Volume With Your Buttons
7 Maintainer: Litten <hello@litten.sh>
8 Author: Litten <hello@litten.sh>
9 Section: Tweaks
```

If you add your email like I did the user will be able to directly contact you by email if they tap on the Author's name

Section 4.2 - Modifying The Makefile

I recommend you to make your Makefile look like mine

THINGS I ADDED:

ARCHS tells theos to compile for the given architectures like arm64 for non A12 64bit devices and arm64e for A12 devices

TARGET tells theos which SDK should be used for compiling, I will always use the latest sdk I have to have the newest features

\$(TWEAK_NAME)_FRAMEWORKS defines which frameworks will be used, you will mostly work with the UIKit so always add it (you don't need to add it, because it's being added automatically but I always do it anyway)

I also changed FeelTheAudio_Files to \$(TWEAK_NAME) to not have to always write the name of the tweak, some basic bash thing

```
M Makefile X
M Makefile
1  ARCHS = arm64 arm64e
2  TARGET = iphone:clang:13.2:13.2
3
4  INSTALL_TARGET_PROCESSES = SpringBoard
5
6  include $(THEOS)/makefiles/common.mk
7
8  TWEAK_NAME = FeelTheAudio
9  $(TWEAK_NAME)_FILES = Tweak.x
10 $(TWEAK_NAME)_CFLAGS = -fobjc-arc
11 $(TWEAK_NAME)_FRAMEWORKS = UIKit
12
13 include $(THEOS_MAKE_PATH)/tweak.mk
14
```

CFLAGS is for memory management, don't remove it

So here's a quote from @qwertyuiop1379 about the \$(TWEAK_NAME):

"You're not supposed to use \$(TWEAK_NAME) in the makefile."

qwertyuiop1379 Developer · 8 points · 5 hours ago

It's actually a list, not a single variable. Similar to the ARCHS variable. If you set that like so:

```
ARCHS = arm64 arm64e
```

then writing

```
$(ARCHS)_Something
```

would expand to

```
arm64 arm64e_Something
```

and cause an error. This means that you can have multiple tweaks by doing:

```
TWEAK_NAME = Tweak1 Tweak2
Tweak1_... = ...
Tweak2_... = ...
```

and using \$(TWEAK_NAME) would break it.

Reply Give Award Share Report Save

The explanation for this is simply because \$(TWEAK_NAME) is a list variable

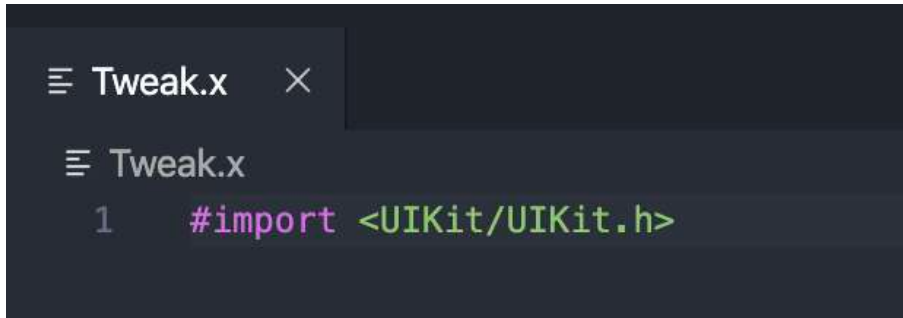
That doesn't mean you're not allowed to do it like I do, and I won't change my way of doing it

Section 5 - Creating Our First Tweak

In this section we're going to make our tweak do something so jump into the tweak.x file to get started

First of all delete everything in the file to get ready

As we use the UIKit import it by adding `#import <UIKit/UIKit.h>`, that's how you import frameworks, to add local headers of the tweak just do it like this: `#import "fileName.h"` but I'll do it within one file in this guide to keep it simple



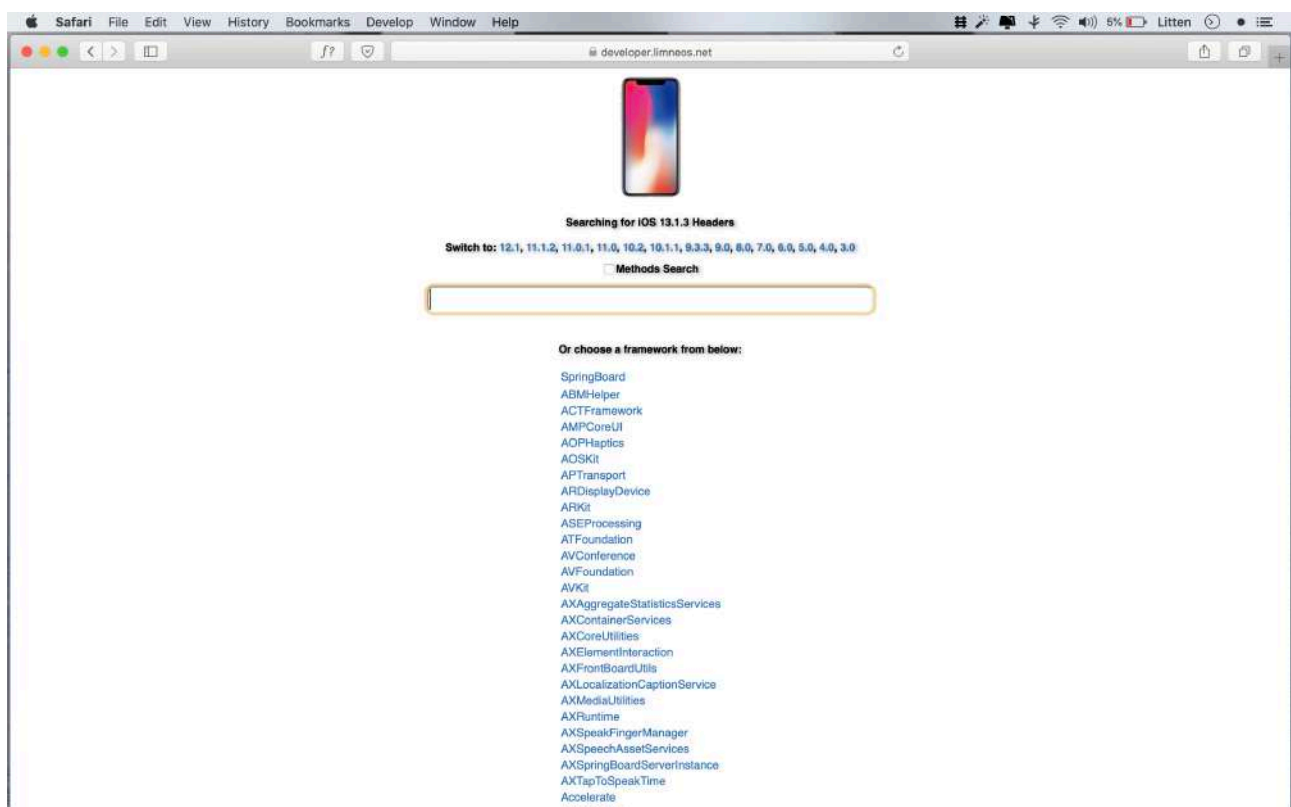
```
1  #import <UIKit/UIKit.h>
```

So now that we imported the UIKit we can add the functionality for the haptics

The best way to find methods or classes is to use Limneos's header website to discover iOS headers and methods <https://developer.limneos.net>

So now click on the link or open that website because I'll show you how to find what we're going to need

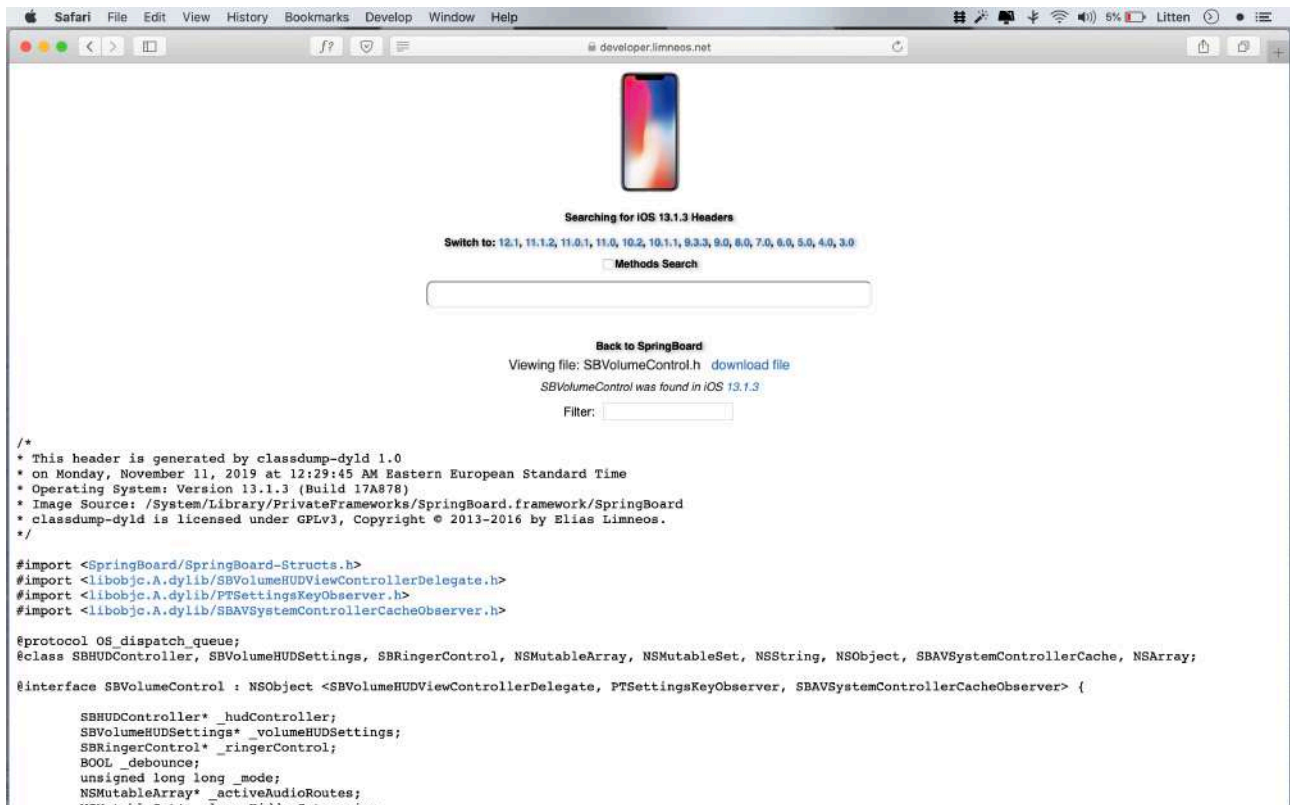
This is how the website will look like:



We're looking for a header so we don't tick the "Methods Search" box

Put SBVolumeControl in the search bar and it will find the header for you, then click on it

SBVolumeControl is only available on iOS 13 so the tweak won't work on iOS 12 or lower, to fix that do the same what we're doing next but hook VolumeControl as well, VolumeControl uses the same methods so just copy the next steps (also don't forget to update the necessary firmware version if using)



This is how it will look like, here we can view all properties and methods of that specific class but we will only need these two:

```
-(void)increaseVolume;  
-(void)decreaseVolume;
```

These two functions get called when the user presses one of the volume buttons, increase for Volume Up and decrease for Volume Down

Now let's hook the SBVolumeControl to modify these two methods

To hook a class means as much as grabbing a basket with apples in it and then adding or removing some of them

It's the same with our code, we hook/grab the class and instead of modifying the amount of apples we're going to modify the code functionality

```
≡ Tweak.x ×
≡ Tweak.x
1  #import <UIKit/UIKit.h>
2
3  %hook SBVolumeControl
4
5
6
7  %end
```

Now that we hooked the class, we're going to change the functionality of the increaseVolume and decreaseVolume methods

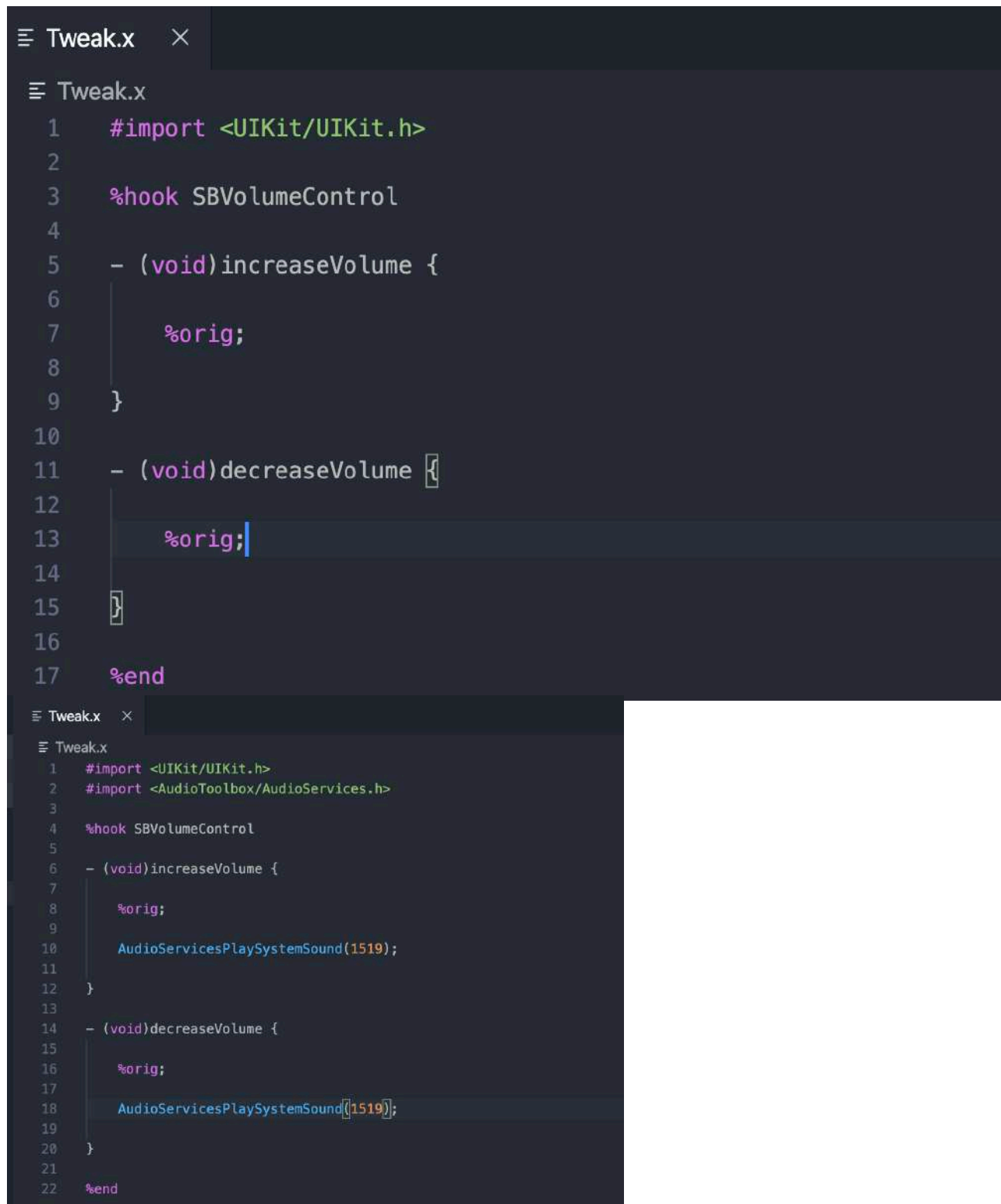
We do it like this:

```
≡ Tweak.x ×
≡ Tweak.x
1  #import <UIKit/UIKit.h>
2
3  %hook SBVolumeControl
4
5  - (void)increaseVolume {
6
7
8
9  }
10
11 - (void)decreaseVolume {
12
13
14
15 }
16
17 %end
```

It's important to know that when we modify a method we want it to continue executing the original code by Apple so add `%orig;` to make it do that

What if we don't add it? Pretty simple, if we don't add it to our volume method we won't be able to change the volume with our buttons anymore until we update or uninstall the tweak because Apple's code doesn't get executed anymore

So just always keep track of the original code



```
≡ Tweak.x ×
≡ Tweak.x
1  #import <UIKit/UIKit.h>
2
3  %hook SBVolumeControl
4
5  - (void)increaseVolume {
6
7      %orig;
8
9  }
10
11 - (void)decreaseVolume {
12
13     %orig;
14
15 }
16
17 %end

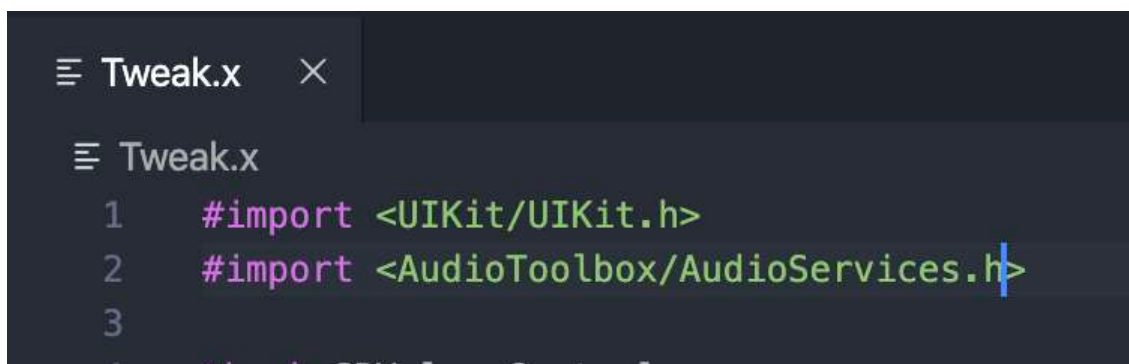
≡ Tweak.x ×
≡ Tweak.x
1  #import <UIKit/UIKit.h>
2  #import <AudioToolbox/AudioServices.h>
3
4  %hook SBVolumeControl
5
6  - (void)increaseVolume {
7
8      %orig;
9
10     AudioServicesPlaySystemSound(1519);
11
12 }
13
14 - (void)decreaseVolume {
15
16     %orig;
17
18     AudioServicesPlaySystemSound(1519);
19
20 }
21
22 %end
```

At this point we can add our haptic feedback as we got everything else set up

To make use of our Haptic/Taptic Engine we need to import another framework in our tweak.x and makefile

Add **AudioToolbox** to your frameworks in the makefile and **#import <AudioToolbox/AudioServices.h>** to your tweak.x

```
8  TWEAK_NAME = FeelTheAudio
9  $(TWEAK_NAME)_FILES = Tweak.x
10 $(TWEAK_NAME)_CFLAGS = -fobjc-arc
11 $(TWEAK_NAME)_FRAMEWORKS = UIKit AudioToolbox
12
```



```
Tweak.x
Tweak.x
1  #import <UIKit/UIKit.h>
2  #import <AudioToolbox/AudioServices.h>
3
```

To actually make our phones vibrate we'll use the **AudioServicesPlaySystemSound** method which is part of the AudioServices.h we just imported

- **AudioServicesPlaySystemSound(1519);** is light feedback
- **AudioServicesPlaySystemSound(1520);** is medium feedback
- **AudioServicesPlaySystemSound(1521);** is strong feedback

The method "AudioServicesPlaySystemSound" is part of Objective-C, these integers (numbers) are specified to "play" haptics, just google it, I don't know why it's those numbers

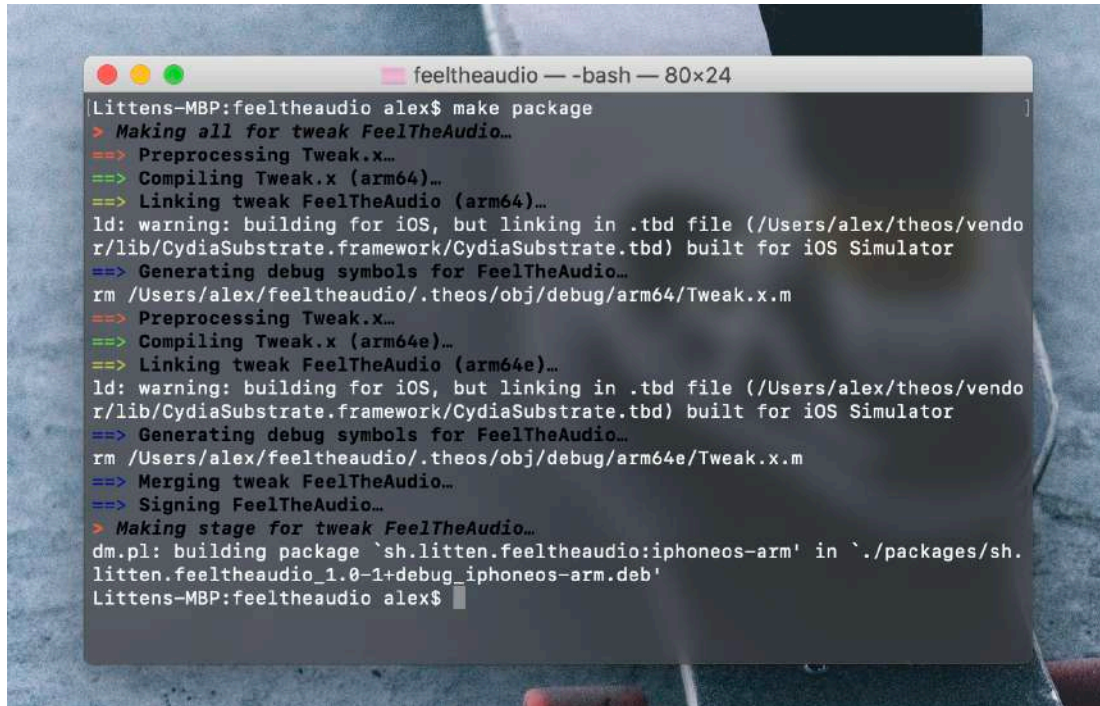
Now let's compile and test it~~

To do that cd into your tweaks directory and type **make package**

You should see this:

Alternatively you could type **make package install** to directly install it on your phone, but you'd need to add a new variable to the makefile to be able to do so

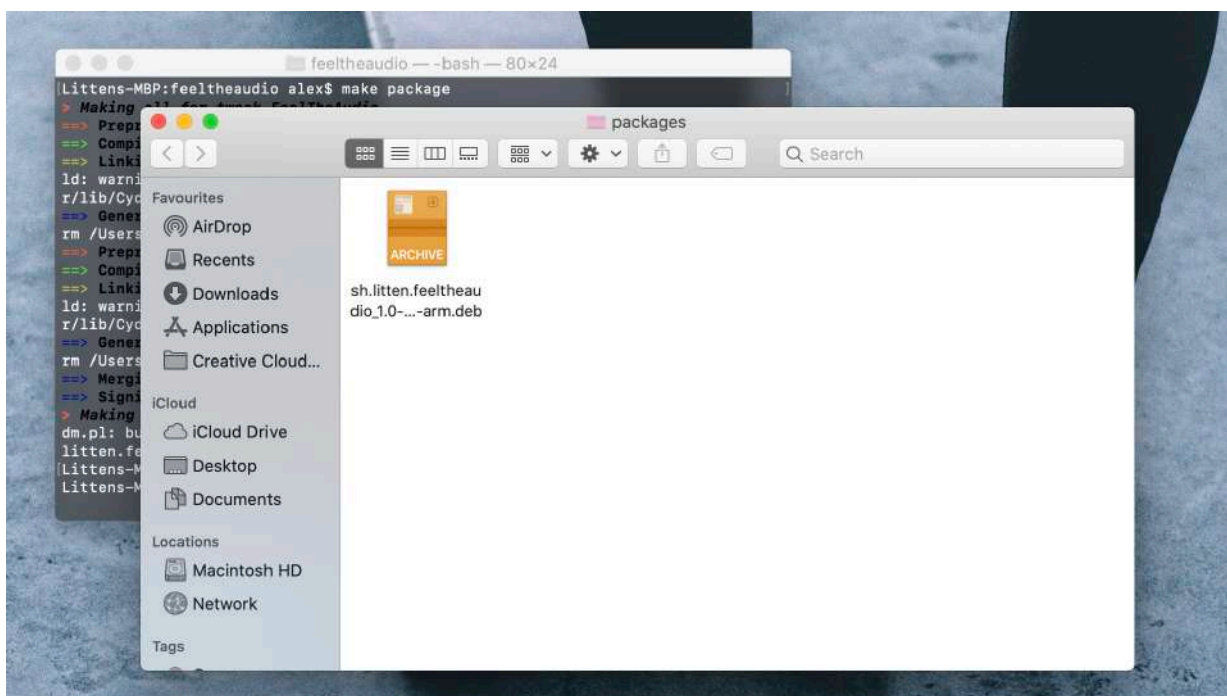
Add this to the makefile to use make package install: THEOS_DEVICE_IP = yourPhonesIP



```
Littens-MBP:feeltheaudio alex$ make package
> Making all for tweak FeelTheAudio...
==> Preprocessing Tweak.x...
==> Compiling Tweak.x (arm64)...
==> Linking tweak FeelTheAudio (arm64)...
ld: warning: building for iOS, but linking in .tbd file (/Users/alex/theos/vendor/lib/CydiaSubstrate.framework/CydiaSubstrate.tbd) built for iOS Simulator
==> Generating debug symbols for FeelTheAudio...
rm /Users/alex/feeltheaudio/.theos/obj/debug/arm64/Tweak.x.m
==> Preprocessing Tweak.x...
==> Compiling Tweak.x (arm64e)...
==> Linking tweak FeelTheAudio (arm64e)...
ld: warning: building for iOS, but linking in .tbd file (/Users/alex/theos/vendor/lib/CydiaSubstrate.framework/CydiaSubstrate.tbd) built for iOS Simulator
==> Generating debug symbols for FeelTheAudio...
rm /Users/alex/feeltheaudio/.theos/obj/debug/arm64e/Tweak.x.m
==> Merging tweak FeelTheAudio...
==> Signing FeelTheAudio...
> Making stage for tweak FeelTheAudio...
dm.pl: building package `sh.litten.feeltheaudio:iphoneos-arm' in `./packages/sh.litten.feeltheaudio_1.0-1+debug_iphoneos-arm.deb'
Littens-MBP:feeltheaudio alex$
```

Congrats on your first tweak!~

Now just open the new folder which got generated and airdrop (or send it somehow else like make package install) to yourself and install it with Filza or Zebra and respring



We will come back to the haptic volume tweak later on to add more reliable haptics as well as preferences (cephai used), but now we're looking at some new method with a new tweak we'll create ^^

Section 6 - Layout Subviews

Using the layoutSubviews is one way, there are also other ways, but it should be kept simple

Basically the layout subviews is/are the view or look of something, let me give you an example:

The UISwitch (known for our lovely switches) has of course also a method called layoutSubviews like every iOS element, if you would modify the layoutSubviews without calling the %orig, the switch would basically be gone, because the look, view simply doesn't exist so to say

And what we're going to create now is to color the UISwitch's on state color (the color when you turn on a switch)

First step is to set up a new theos project, as always give it a name, identifier, etc and apply the same steps we did for the control and makefile (again we are targeting the UIKit so make sure to have com.apple.UIKit instead of com.apple.springboard in the plist)

I have also set up a new project called SwitchTheColors

```
control
control
1 Package: sh.litten.switchthecolors
2 Name: SwitchTheColors
3 Depends: mobilesubstrate
4 Version: 1.0
5 Architecture: iphoneos-arm
6 Description: Change the color of your Switches!
7 Maintainer: Litten <hello@litten.sh>
8 Author: Litten <hello@litten.sh>
9 Section: Tweaks
10
```

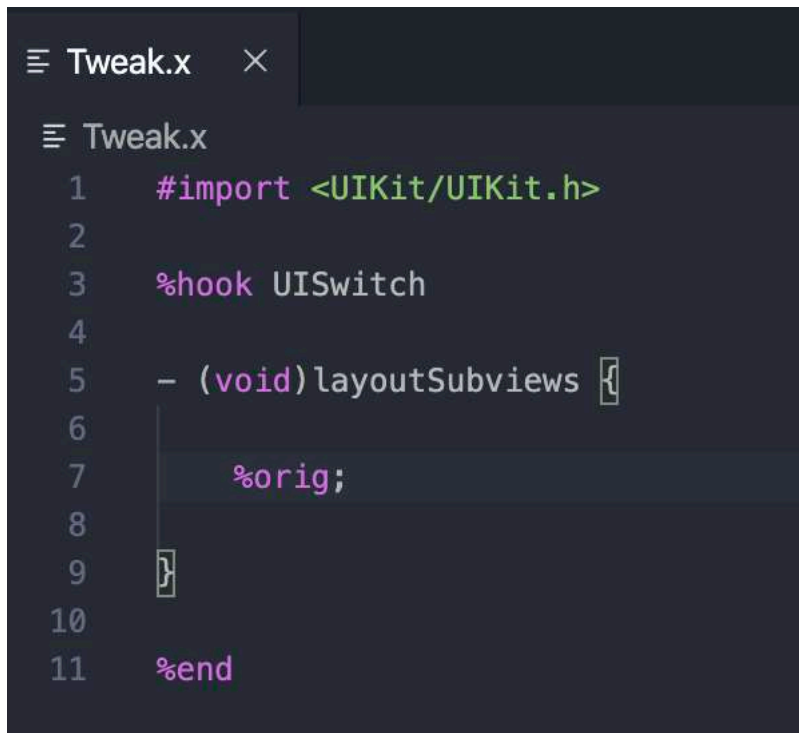
```
Makefile
Makefile
1 ARCHS = arm64 arm64e
2 TARGET = iphone:clang:13.2:13.2
3
4 INSTALL_TARGET_PROCESSES = SpringBoard
5
6 include $(THEOS)/makefiles/common.mk
7
8 TWEAK_NAME = SwitchTheColors
9 $(TWEAK_NAME)_FILES = Tweak.x
10 $(TWEAK_NAME)_CFLAGS = -fobjc-arc
11 $(TWEAK_NAME)_FRAMEWORKS = UIKit
12
13 include $(THEOS_MAKE_PATH)/tweak.mk
14
```

```
SwitchTheColors.plist
SwitchTheColors.plist
1 { Filter = { Bundles = ( "com.apple.UIKit" ); }; }
2
```

```
Tweak.x
Tweak.x
1 #import <UIKit/UIKit.h>
```

Because our tweak should modify the UISwitch system wide we will **hook the UISwitch** itself and then **change the functionality of the layoutSubviews method**

If you didn't know it yet, - (void) is how a method is being declared in objective c, a void returns nothing



```
≡ Tweak.x  ×
≡ Tweak.x
1  #import <UIKit/UIKit.h>
2
3  %hook UISwitch
4
5  - (void)layoutSubviews {
6
7      %orig;
8
9  }
10
11 %end
```

This is how it should look like for now before getting started, **don't forget the %orig;**

So this is how we'll change the color:

A UISwitch has a method called **setOnTintColor**, we can easily call it within the layoutSubviews because the layoutSubviews are being called very often

To call a function in objective c we just type: [self methodName];

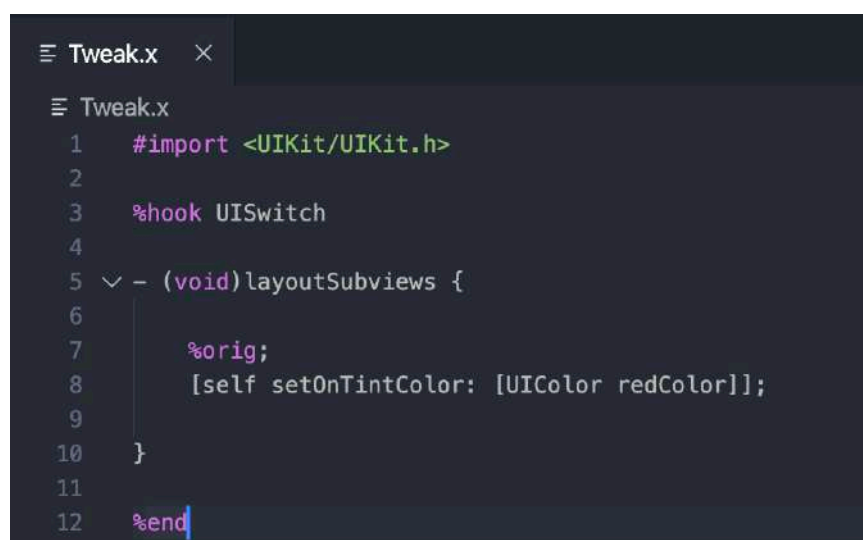
We will do that for the UISwitch now, [self setOnTintColor] but as the **setOnTintColor** wants us to also specify a value we'll need to do it like this: [self setOnTintColor: [UIColor redColor]];

We use self because we assign the color to the object we're targeting itself

UIColor redColor is also a method so we need to put the [] around it

It tells the **setOnTintColor** to use a red color because of **UIColor redColor**

You can also use RGBA color like this: [UIColor colorWithRed:1.00 green:0.80 blue:1.00 alpha:1.0]; which would be a light pink

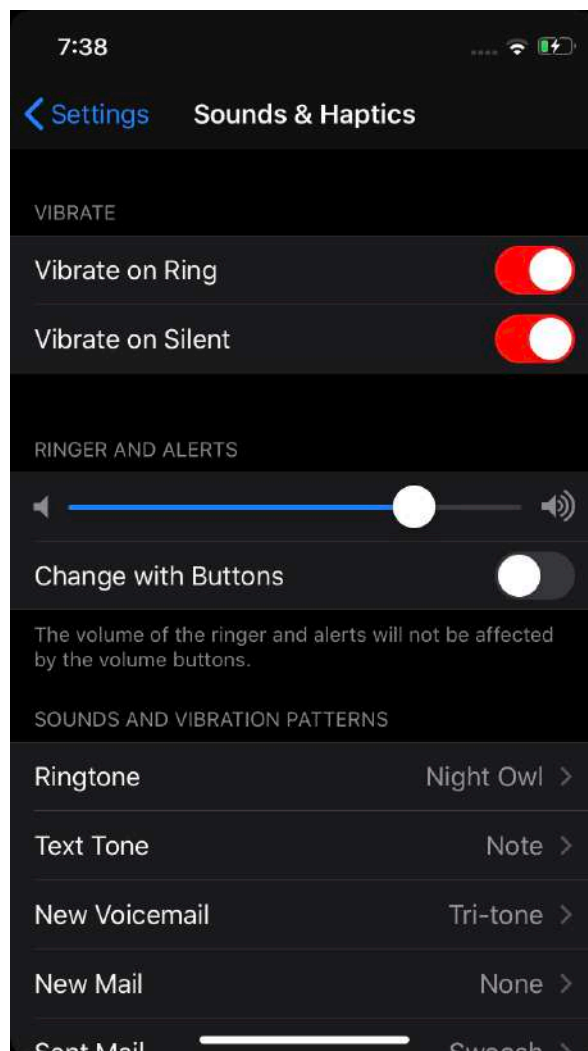


```
≡ Tweak.x  ×
≡ Tweak.x
1  #import <UIKit/UIKit.h>
2
3  %hook UISwitch
4
5  - (void)layoutSubviews {
6
7      %orig;
8      [self setOnTintColor: [UIColor redColor]];
9  }
10
11
12 %end
```

Another very simple but cool little tweak ^-^

```
switchthecolors — -bash — 80x24
[Littens-MBP:~ alex$ cd switchthecolors/
[Littens-MBP:switchthecolors alex$ code .
[Littens-MBP:switchthecolors alex$ make package
> Making all for tweak SwitchTheColors...
==> Preprocessing Tweak.x...
==> Compiling Tweak.x (arm64)...
==> Linking tweak SwitchTheColors (arm64)...
ld: warning: building for iOS, but linking in .tbd file (/Users/alex/theos/vendor/lib/CydiaSubstrate.framework/CydiaSubstrate.tbd) built for iOS Simulator
==> Generating debug symbols for SwitchTheColors...
rm /Users/alex/switchthecolors/.theos/obj/debug/arm64/Tweak.x.m
==> Preprocessing Tweak.x...
==> Compiling Tweak.x (arm64e)...
==> Linking tweak SwitchTheColors (arm64e)...
ld: warning: building for iOS, but linking in .tbd file (/Users/alex/theos/vendor/lib/CydiaSubstrate.framework/CydiaSubstrate.tbd) built for iOS Simulator
==> Generating debug symbols for SwitchTheColors...
rm /Users/alex/switchthecolors/.theos/obj/debug/arm64e/Tweak.x.m
==> Merging tweak SwitchTheColors...
==> Signing SwitchTheColors...
> Making stage for tweak SwitchTheColors...
dm.pl: building package `sh.litten.switchthecolors:iphoneos-arm' in `./packages/
sh.litten.switchthecolors_1.0-1+debug_iphoneos-arm.deb'
Littens-MBP:switchthecolors alex$
```

Again, install the deb and respring and you will see the changes 🤗



Section 7 - Extension For Our Haptic Volume Tweak

Coming next

Side Section - Creating Your Own Repo To Host Tweaks And Apps

I've created a little template for you to use, my personal repo has a lot more files but it should be an easy understandable template

At first download my template here: <https://litten.sh/OwO/Repository.zip> and extract it

pt1 Getting The Repo Files Ready:

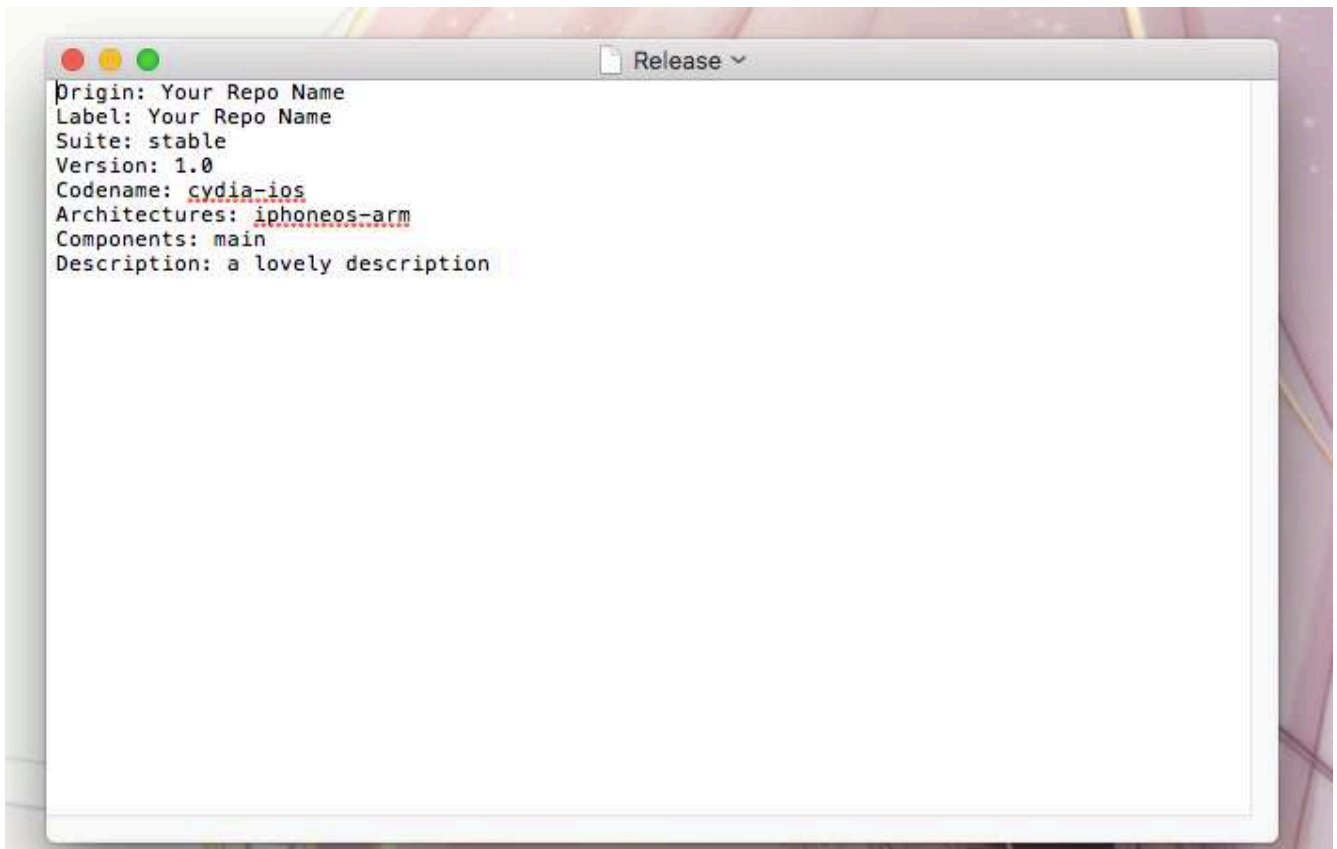
So you should have 5 files and one folder now that you've downloaded my template, let's correct everything

For the icon (called [Cydialcon.png](#), it must be named like this) you can use any image, optionally you can copy paste that image two times so you got it three times, then just rename the second one to [Cydialcon@2x.png](#) and the third one to [Cydialcon@3x.png](#), that's for resizing on different screen sizes

The next file on our list is the Release file:

Here you can edit the name which is being shown in any package manager and the description for it

Just open the release file with any text editor and change the origin, label and description to fit your information



Now we're almost done, just drag and drop your Tweak (.deb) into the debs folder

Then open a terminal, cd into your repo directory and type `./scan-packages.sh` which will automatically generate all information and hashes about the deb file and store it in the Packages file with the provided script **Now go ahead, upload your repo (next pt/s) and try it out** 🤖

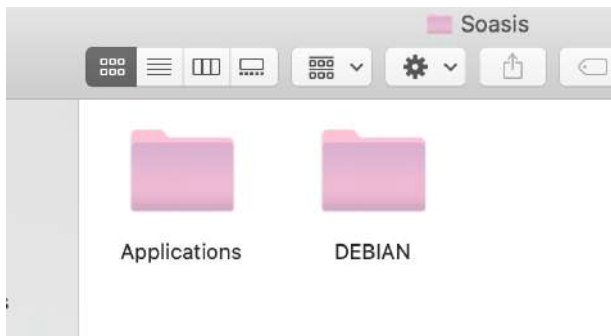
But what if you want to host an app like an IPA? It's easy:

Get your IPA and rename it to .zip (basically just extract it) and copy .app from it

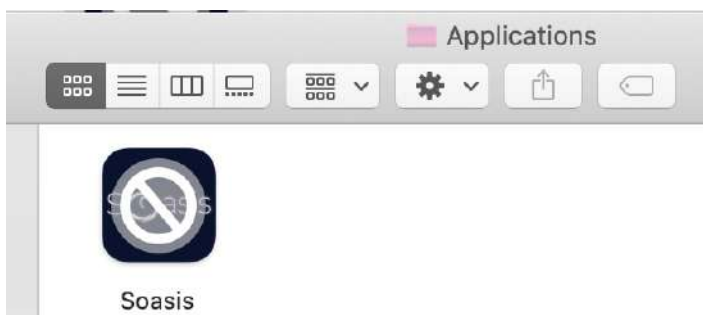


Now that you got the .app create a new folder with the name of the app, navigate into it and create two folders, one called DEBIAN and one called Applications, the .app file belongs to the Applications folder

Then copy the control file of one of your tweaks and paste it into the DEBIAN folder and again edit the information in the control file



The last step is to open a terminal window and type `dpkg -b "drag your folder in"`, this will give you a deb file which you can simply add to the debs folder of your repo and upload it, done, your app is now downloadable from your repo

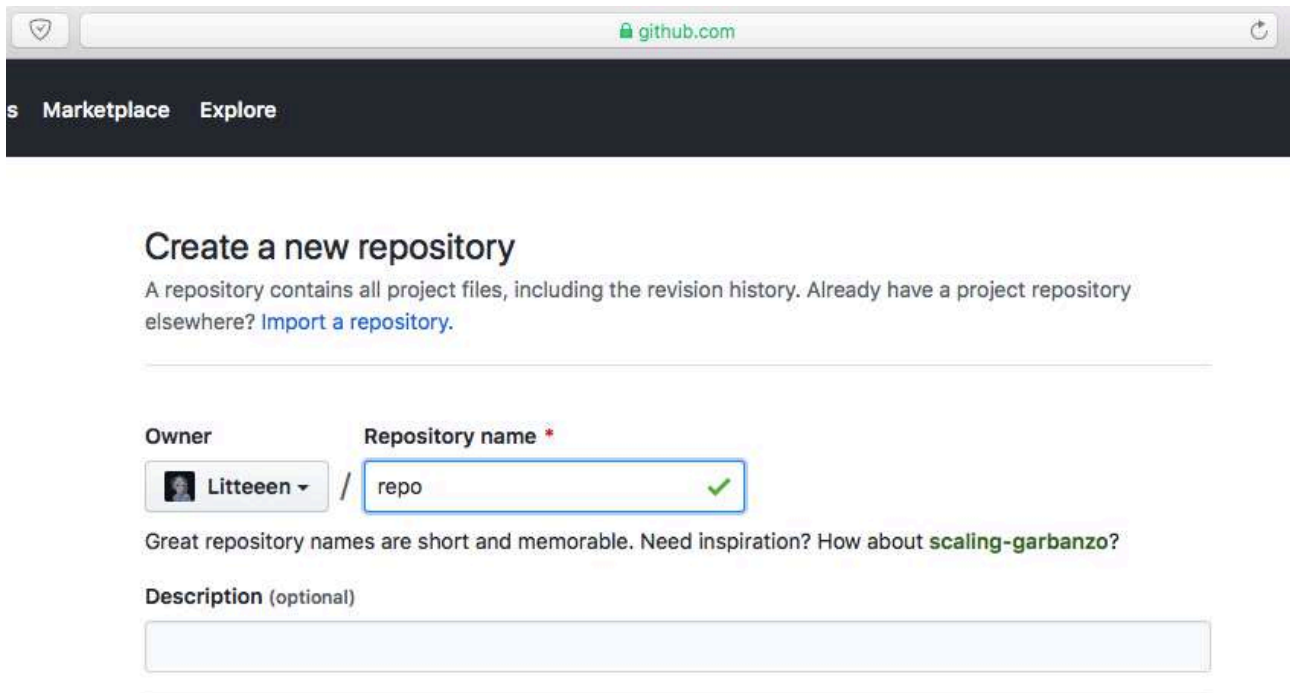


pt2 Your Own Server:

If you have your own server then you can just upload the repo folder and add the repo (link to the repo folder) in your package manager

pt3 Using Github Pages:

First step is to create a new repository on GitHub, you can call it whatever you want but I'll name it repo



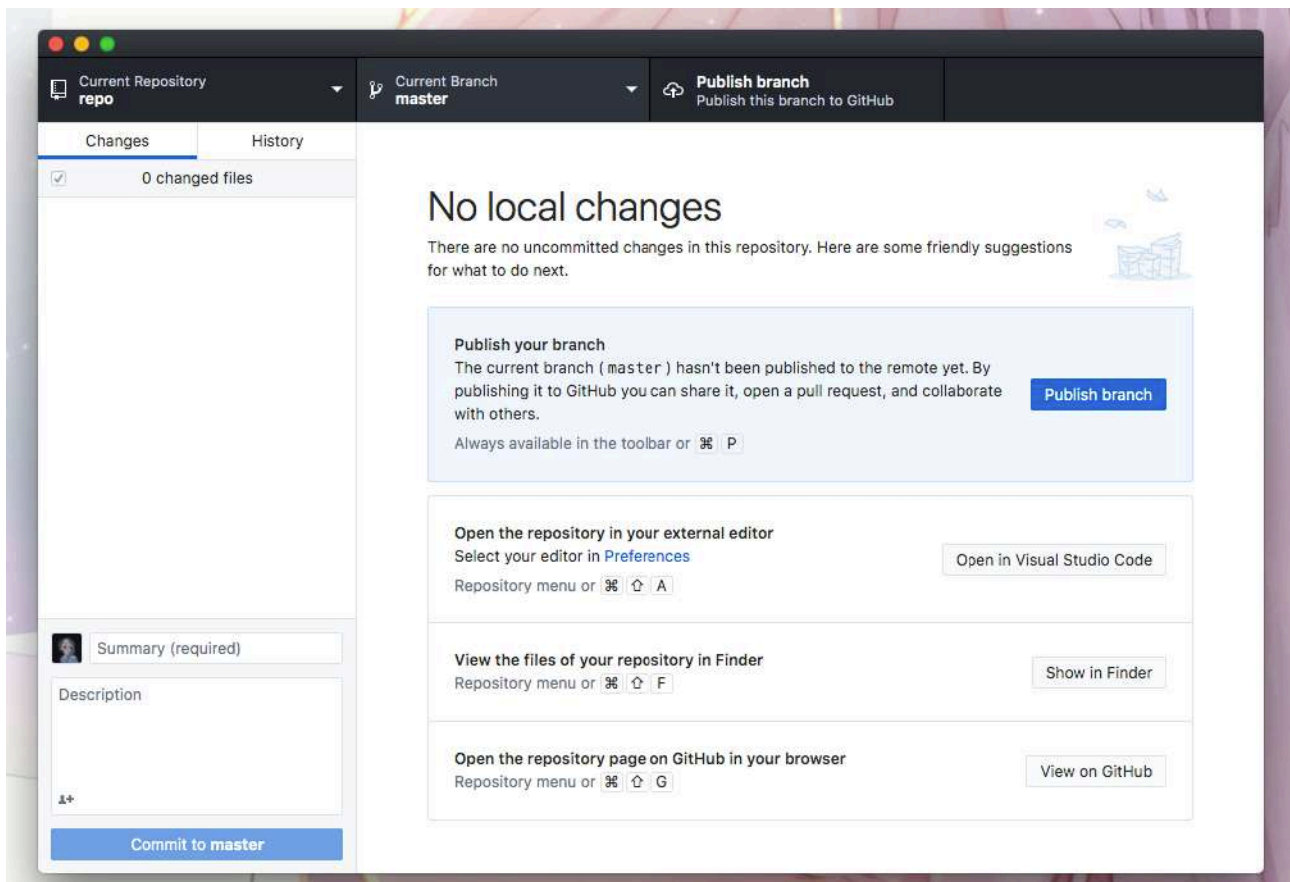
The screenshot shows the GitHub web interface for creating a new repository. The browser address bar shows 'github.com'. The navigation bar includes 'Marketplace' and 'Explore'. The main heading is 'Create a new repository'. Below it, a subtext explains that a repository contains all project files and revision history, with a link to 'Import a repository'. The form has two main sections: 'Owner' and 'Repository name'. The 'Owner' dropdown is set to 'Litteen'. The 'Repository name' text input contains 'repo' and has a green checkmark icon to its right. Below this, a tip suggests repository names should be short and memorable, with an example 'scaling-garbanzo?'. There is also a 'Description (optional)' text area.

It's easiest to use Github Desktop (install Github Desktop before continuing)

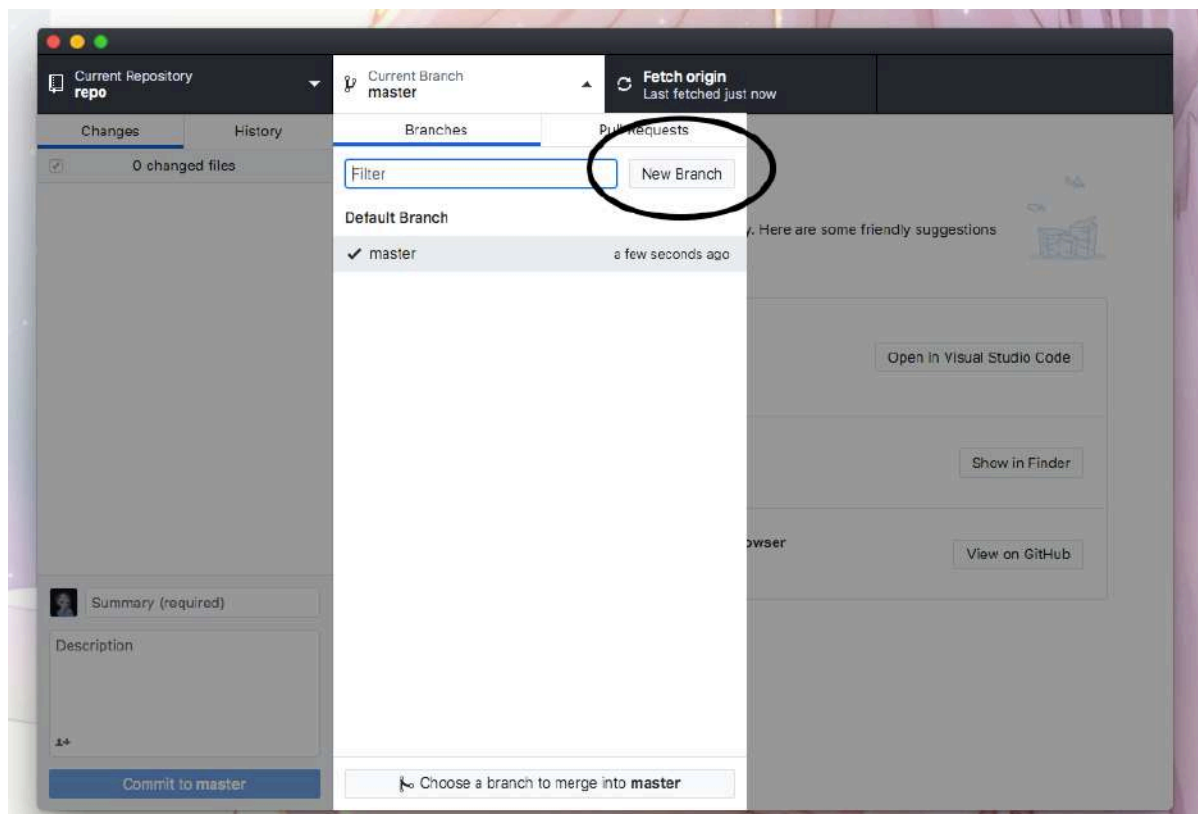


This screenshot shows the 'Quick setup' section of the GitHub repository creation process. The title is 'Quick setup — if you've done this kind of thing before'. Below the title, there are three buttons: 'Set up in Desktop', 'HTTPS', and 'SSH'. The 'Set up in Desktop' button is circled in black. To the right of these buttons is a text input field containing the SSH URL 'https://github.com/Litteen/repo.git'. Below the buttons and input field, there is a line of text: 'Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.'

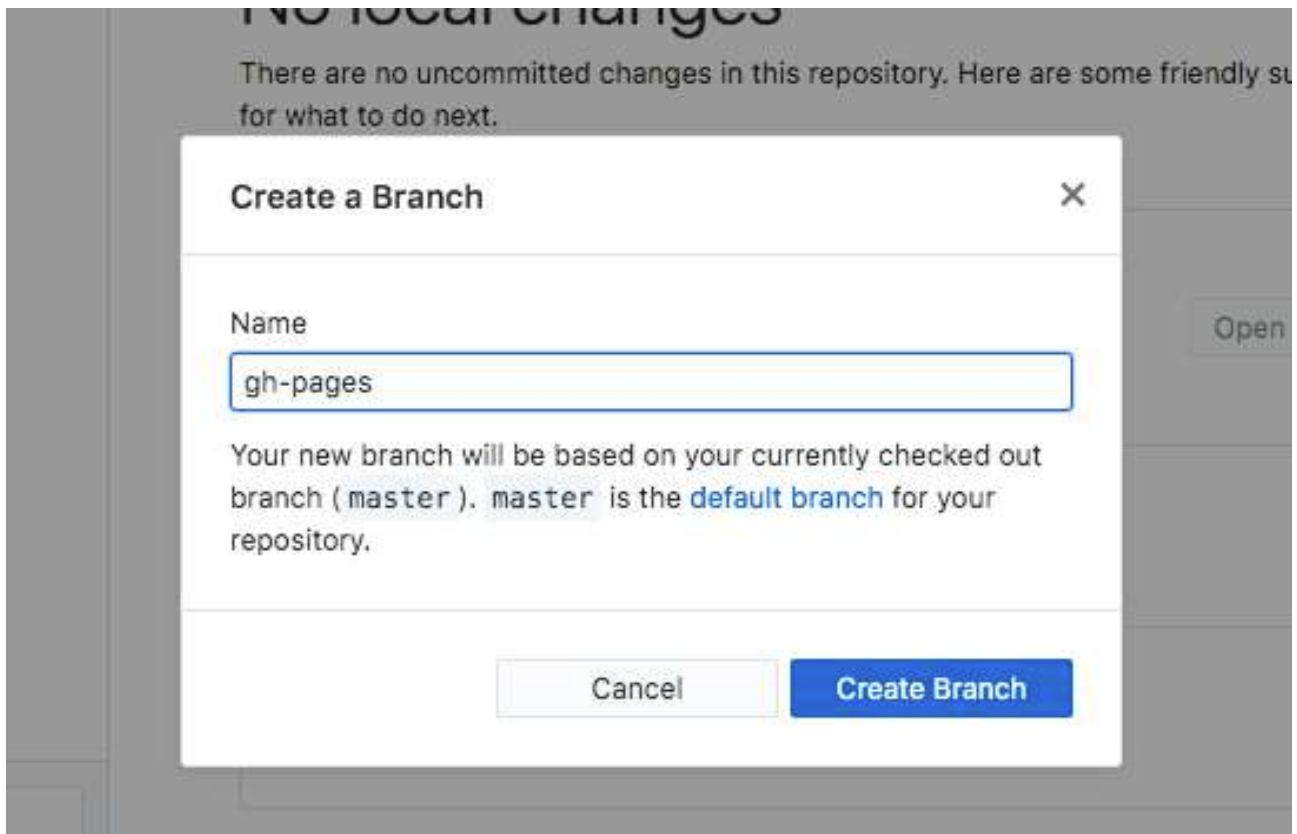
Now if you have your repo ready to be uploaded put the repo files into the GitHub repository folder and publish it



After publishing to the master branch we need to create a new branch called gh-pages so click on "Current Branch" at the top and then on "New Branch"



Name it **gh-pages** and create it, then publish the branch again



After that switch back to the master branch by clicking Current branch again on the top and navigate to master, optionally refresh once

At this point we're done with the repo, if you got a new tweak just drag it into the deps folder, run the script, update the repo by opening GitHub desktop and publish the new things

The link for your repo will be: yourGithubName.github.io/repoName
For me it would be: litteen.github.io/repo

~Litten 🐱💕