What to do with an old iPad

Sep 5, 2025

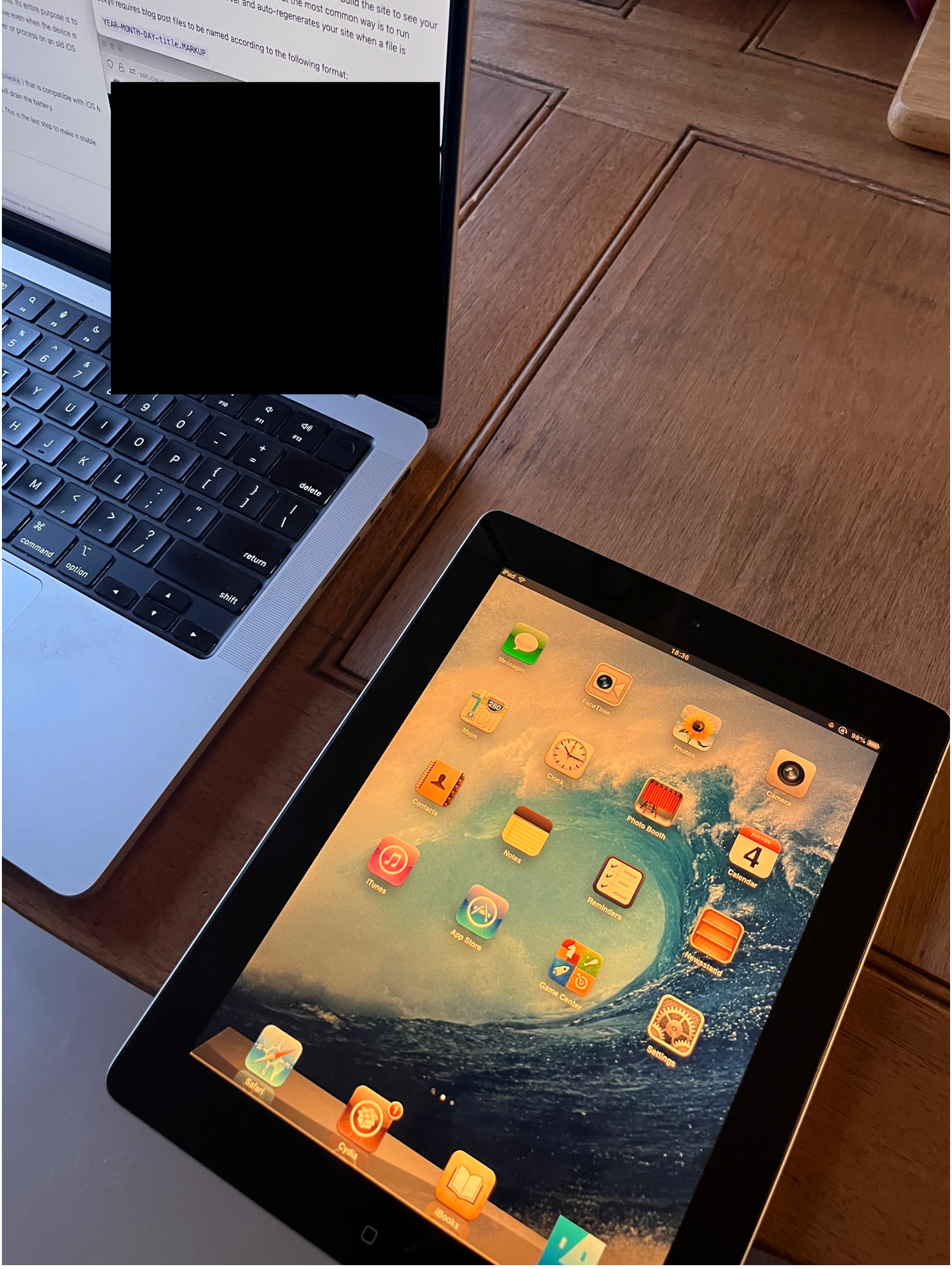
I recently *inherited* my parents' old iPad 2. It had iOS 9 on it and was barely usable, in part due to how slow it was but mostly because of old SSL certificates and apps stopping support. But I wanted to give it some life because I hate to see a working machine go to waste. So I asked around and got a great answer:

> *The iPad 2 can be downgraded to iOS 6.1.3 or 8.4.1. Both offer better performance than iOS 9 and have untethered jailbreaks. Once downgraded and jailbroken you can sideload hours and hours worth of games. A lot from that era are IAP-free.*

So that's where my journey began.

Downgrading

Jailbreaking the iPad 2 is fairly easy thanks to iOS CFW Guide(I didn't even realize I was following the guide for iPad 3, but it worked perfectly nonetheless). Once that is done, you can downgrade using Legacy-iOS-kit which is also pretty straightforward. And that's it: you have your iPad jailbroken and on the software it was meant to run. iOS 6.1.3 is not as useful as some of the newer versions, but it's pretty snappy and very aesthetically pleasing.



But what can you do with it?

A lot. You can download **iFile** and access files on device, something that was not possible on iOS 6. So that basically means you can load and watch any kind of media on the device. You can torrent files if you want with **iTransmission** (remember when everything started with a lowercase i?). Load up some epubs and pdfs and read them on iBooks. Get **f.lux** on device to get warm light on screen. Download some old iOS games.

But that's pretty boring. I wanted to do more with it and was unsatisfied with my big-ass reader. One thing that I didn't mention is that when you jailbreak, you get root access to the device. That means you can ssh into it (which you do for the downgrade) and use the unix terminal to do whatever[1] you want. So I thought, what if I use this as a tool to learn a bit more about hosting/networking?

Hosting on iPad

My general idea when I began was:

1. Get a server utility
2. Load a static site into the iPad
3. Tunnel out
4. ???
5. Profit

1. Server

Since the hardware is old, I needed a light web server that could run on very little resources. Enter lighttpd (pronounced *lighty* I'm told by Gemini). And since there was no package manager available on the iPad's terminal, I had to load it from *Cydia* (like an appstore for tweaks that you get once you jailbreak). Easy. Done.

2. Static Site

I had no idea what I was going to host, but it was irrelevant anyway since the site itself was not the purpose of the project. So I got Jekyll installed on my pc, ran the `jekyll build` command, and loaded the HTML + CSS files on the iPad. Another step done. At this point I had the server up and running locally, serving the blog on the local network. Pretty good already,

3. Tunnel out

I knew *tunneling* was an option for self hosting, but I didn't know what it really was. Tunneling, as I understand it, is setting an SSH connection to a server which processes requests, and forwards them to your local machine, in this case the iPad. It's supposed to be *safer* than **port forwarding** since you're not exposing your home's IP this way. The most common provider is cloudflare, but you need to install `cloudflared` on the machine you're using to host, so that was a no-go (a Cydia search for *cloudflare* yielded exactly 0 results).

Enter *localhost.run*, or the first wall I hit. LHR allows you to ssh into it without needing any special software on your machine, connecting a tunnel to your web app running locally. It seems very easy at first, and I guess if you're doing it on a regular computer it is, but it frustrated the hell out of me. First, the encryption algorithms on iOS 6.1.3 are very old, and that proves a challenge when you're trying to SSH into anything. I figured this out by adding a flag to the `ssh` command to accept `rsa` encryption. It seemed to work, but wouldn't recognize my user, so I went and created one. Upon account creation I realized that I needed to pay $9 monthly to set a custom domain, otherwise LHR assigns a random domain which changes periodically. Paying for it kinda defeated the purpose, but I decided to try the free version anyway. It didn't work. On to the next attempt.

So if pre-built solutions don't work what do you do? You build your own, of course! The idea was to get a free Virtual Private Server from Google Cloud Platform, get `nginx` to handle the requests, set up a custom tunnel between VPS and iPad, and serve requests from `lighttpd`. Sounds very simple, however, I ran into a problem almost instantly: there was no way to `autossh` (same as ssh, but reconnects if the connection drops) from the iPad into the VPS! Since iOS 6 there have been many advances in cryptography it seems, and due to the ancient OpenSSL version, the VPS wouldn't allow the connection. After much debugging, Gemini summarized the issue perfectly:

> *The server receives the signature from your iPad, but it rejects it. Your iPad's ancient crypto library (OpenSSL 0.9.8zg from 2015) is creating a signature that the modern server considers invalid or insecure.*

And so, my workaround had failed. At this point I realize that what I'm doing is kinda pointless and that I could just host the blog on the VPS and call it a day. But I had sunk much time into this and wasn't about to give up. So **port forwarding** time! If I can't connect from the iPad into the VPS, maybe the VPS can connect into the iPad. After all, my computer was perfectly capable of ssh-ing into the iPad. So after much assurance from both Gemini and my tech-wizard uncle that it was *okay* to port forward, I decided to give it a go. I'll omit many details here just in case I'm exposing myself, but basically, I opened up a port to my router, got a dynamic DNS from FreeDNS, and once that was done, I could get the VPS to connect to my iPad remotely. It felt like magic, but again came the problems: `autossh` was not working, only regular ssh using the same flag to accept `rsa` encryption that I had to use on my computer. Key-based login was failing due to a bug in the iPad's SSH server. Password login, however, worked. In came `sshpass`, which enables `autossh` with password-based authentication. Turns out, this didn't work either, because the `autossh` command needed a `-t` (force terminal) flag for the password prompt. With that cleared up, I could use `nohup` to run the `sshpass` command and keep it running in the background. And that's it! My blog was up and running.

So, if you're reading this post right now, it means my server is working, and that this site is being served by an iPad 2 from 2012, running iOS 6.1.3 and Insomnia to keep it connected to Wi-Fi. And you may ask, was it worth it? Hell yeah. It was frustrating and had many more hurdles than I anticipated, but it's really satisfying to know that I managed to achieve my vision. And I learned a lot in the process, as this was my first time using a VPS and really, hosting anything semi-serious. Regarding the blog, I can't promise to post regularly, but if I do so and it becomes something kind of serious, maybe I'll consider moving the hosting to a VPS. Or keep it on the iPad. If it accomplishes the task of serving this website,

maybe it's *okay* for the job after all. *Tools are only as interesting as you make them with your own creations.*

This post is dedicated to my uncle Mariano, my family's designated tech wizard. Hope you enjoy it! Also, shout out to Google Gemini's 1m token context window.

1. *many unix utilities are missing, but some can be loaded from Cydia* ↩

---

owen-blogs

owen-blogs
owen.botto@gmail.com

owen-makes
owen_makes

A blog about tech, creativity and random projects