****

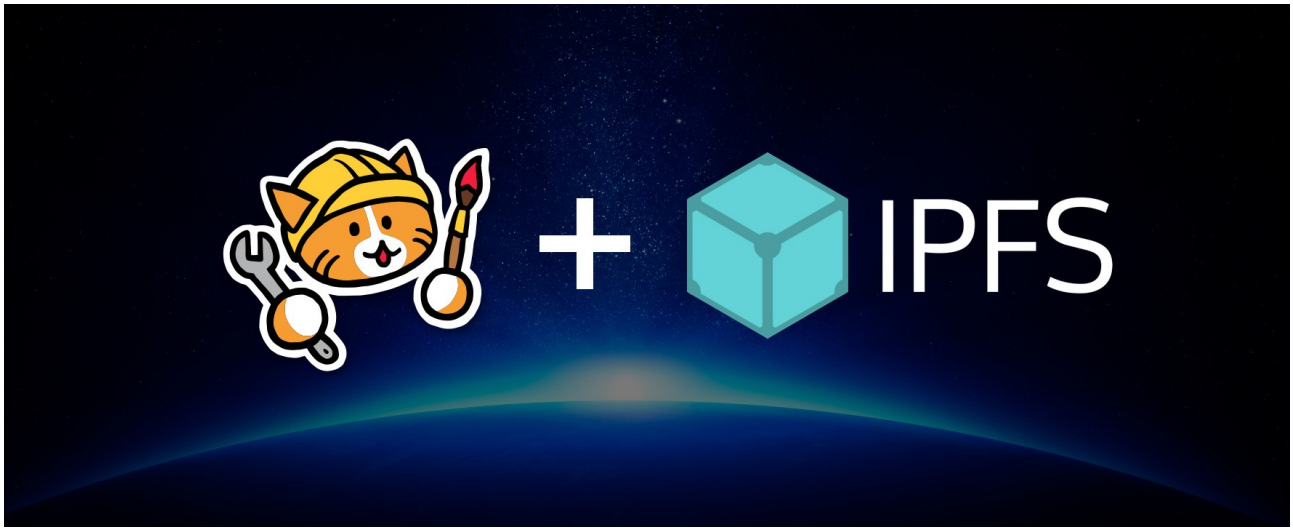
Introduction

if you have internet connections go to the links which mentioned in last page ::

****

HTTP is obsolete. It's time for the distributed, permanent web

September 8th, 2015 - kyledrake



Early this year, the Internet Archive put out a call for a distributed web. We heard them loud and clear.

Today I'm making an announcement that begins our long journey to the future of the web. A web that is faster, more secure, more robust, and more permanent.

Neocities has collaborated with Protocol Labs to become the first major site to implement IPFS in production. Starting today, all Neocities web sites are available for viewing, archiving, and hosting by any IPFS node in the world. When another IPFS node chooses to host a site from Neocities, that version of the site will continue to be available, even if Neocities shuts down or stops hosting it. The more IPFS nodes seed Neocities sites, the more available (and redundant) Neocities sites become. And the less centrally dependent the sites are on us to continue existing.

What is IPFS? From their README:

> **IPFS is a distributed file system that seeks to connect all computing devices with the same system of files.** In some ways, this is similar to the original aims of the Web, but IPFS is actually more similar to a single bittorrent swarm exchanging git objects. IPFS could become a new major subsystem of the internet. If built right, it could complement or replace HTTP. It could complement or replace even more. It sounds crazy. It is crazy.

IPFS is still in the alpha stages of development, so we're calling this an experiment for now. It hasn't replaced our existing site storage (yet). Like with any complex new technology, there's a lot of improvements to make. But IPFS isn't vaporware, it works right now. You can try it out on your own computer, and already can use it to help us serve and persist Neocities sites.

The message I want to send couldn't possibly be more audacious: I strongly believe IPFS *is* the replacement to HTTP (and many other things), and now's the time to start trying it out. Replacing HTTP sounds crazy. It is crazy! But **HTTP is broken**, and the craziest thing we could possibly do is continue to use it forever. We need to apply state-of-the-art computer science to the distribution problem, and design a better protocol for the web.

****

Part 1: What's wrong with HTTP?

****

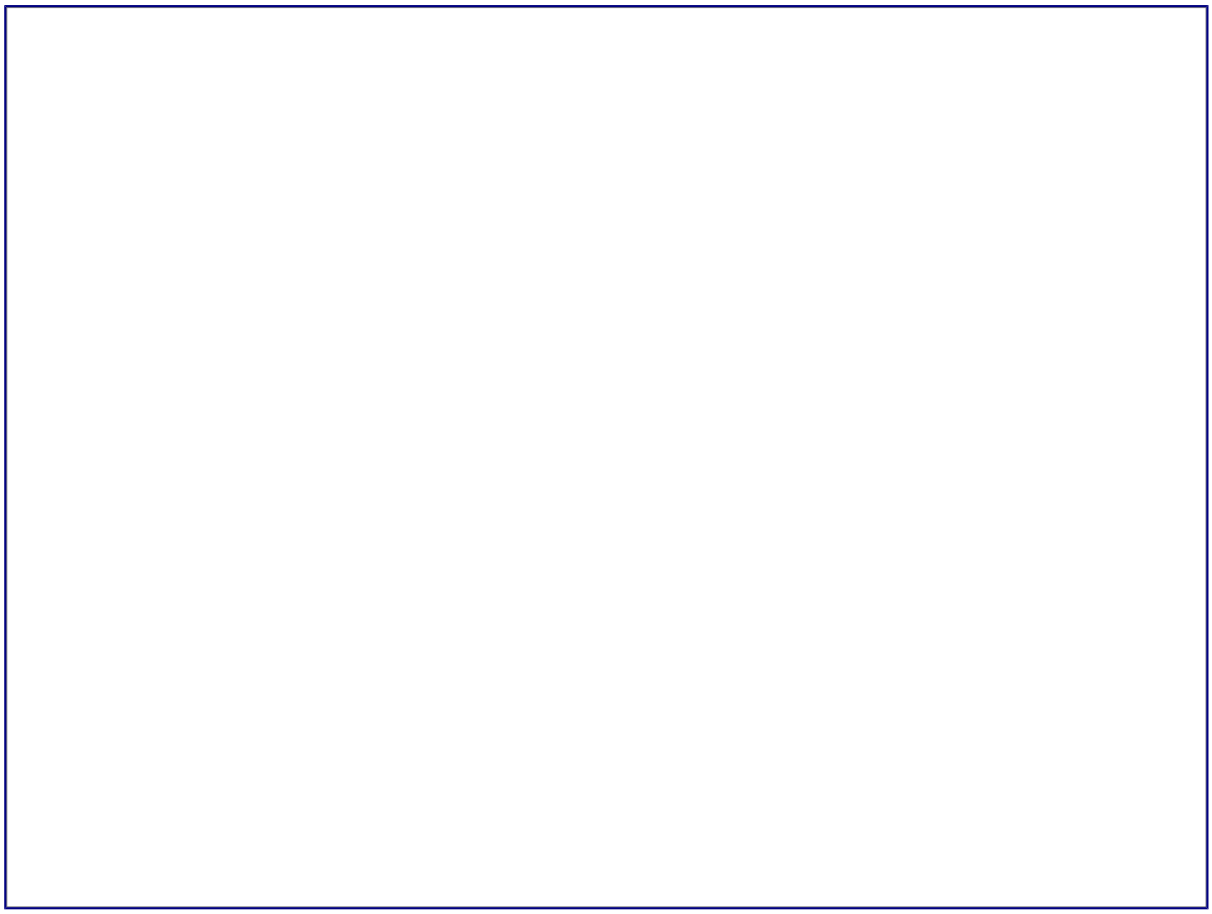
The Hypertext Transfer Protocol (HTTP) has unified the entire world into a single global information protocol, standardizing how we distribute and present information to eachother.

It is inconceivable for me to even think about what life would be like without it. HTTP dropped the cost of publishing content to almost nothing, an innovation that took a sledgehammer to the top-down economic, political, and cultural control over distribution of information (music, ideas, video, news, games, everything). As a result of liquifying information and making it the publication of it more egalitarian and accessible, HTTP has made almost everything about our culture better.

I love HTTP, and I always will. It truly stands among the greatest and most important inventions of all time.

But while HTTP has achieved many things, it's usefulness as a foundation for the distribution and persistence of the sum of human knowledge isn't just showing some cracks, it's crumbling to pieces right in front of us. The way HTTP distributes content is **fundamentally flawed**, and no amount of performance tuneups or forcing broken CA SSL or whatever are going to fix that. HTTP/2 is a welcome improvement, but it's a conservative update to a technology that's beginning to show its age. To have a better future for the web, we need more than a spiced up version of HTTP, we need a new foundation. And per the governance model of cyberspace, that means we need a new protocol. IPFS, I'm strongly hoping, becomes that new protocol.

HTTP is brittle



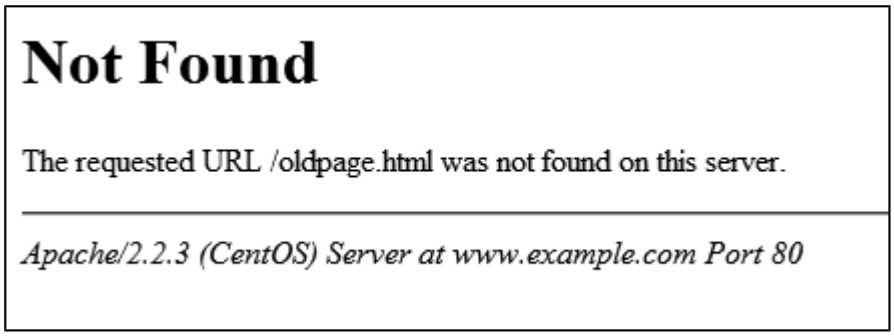
This is a picture of the first HTTP web server in the world. It was Tim Berners-Lee's NeXT computer at CERN.

Pasted on the machine is an ominous sticker: **"This machine is a server, do not power it down!!"**.

The reason it couldn't be powered down is that web sites on other servers were starting to link to it. Once they linked to it, they then depended on that machine continuing to exist. If the machine was powered down, the links stopped working. If the machine failed or was no longer accessible at the same location, a far worse thing happened: the chain between sites becomes permanently broken, and the ability to access that content is lost forever. That sticker perfectly highlights the biggest problem with HTTP: it erodes.

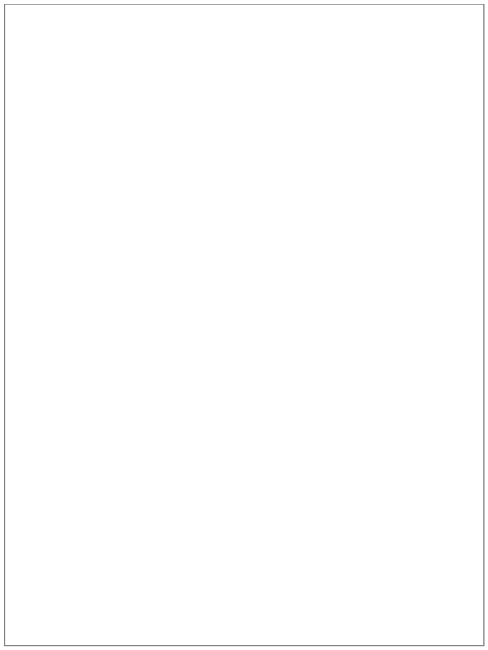
Tim's NeXT cube is now a museum piece. The first of millions of future dead web servers.

You've seen the result:

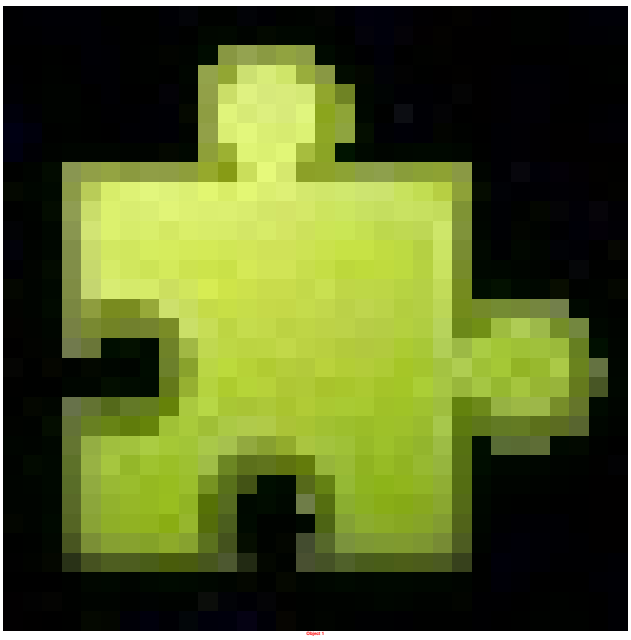


Even if you've never read the HTTP spec, you probably know what 404 means. It's the error code used by HTTP to indicate that the site is no longer on the server at that location. Usually you're not even that lucky. More often, there isn't even a server there anymore to tell you that the content you're looking for is gone, and it has no way to help you find it. And unless the Internet Archive backed it up, you'll never find it again. It becomes lost, forever.

The older a web page is, the more likely it is you'll see 404 pages. They're the cold-hearted digital tombstones of a dying web, betraying nothing about what knowledge, beauty, or irreverent stupidity may have once resided there.



One of my favorite sites from the 90s web was Mosh to Yanni, and viewing the site today gives a very strong example of how inadequate HTTP is for maintaining links between sites. All the static content stored with the site still loads, and my modern browser still renders the page (HTML, unlike HTTP, has excellent lasting power). But any links offsite or to dynamically served content are dead. For every weird example like this, there are countless examples of incredibly useful content that have also long since vanished. Whether eroding content is questionable crap or timelessly useful, it's still our history, and we're losing it fast.



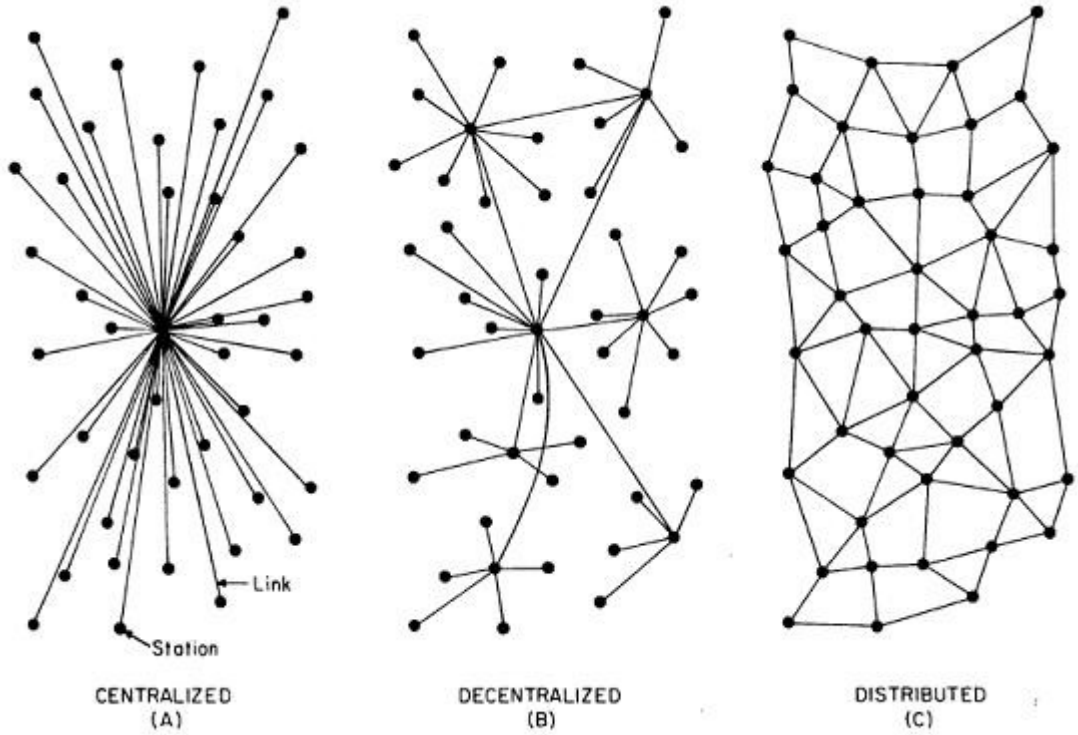
above video url ::https://youtu.be/NdZxI3nFVJs

The reason this happens is simple: **centrally managed web servers inevitably shut down**. The domain changes ownership, or the company that ran it goes out of business. Or the computer crashes, without having a backup to restore the content with. Having everyone run their own personal HTTP server doesn't solve this. If anything, it probably makes it worse.

HTTP encourages hypercentralization

The result of this erosion of data has been further dependence on larger, more organized centralized services. Their short-term availability tends to be (mostly) good due to redundant backups. But this still doesn't address long-term availability, and creates a whole new set of problems.

We've come a long way since John Perry Barlow's A Declaration of the Independence of Cyberspace. As our electronic country becomes more influential and facilitates the world with more information, governments and corporations alike have started to pry into HTTP's flaws, using them to spy on us, monetize us, and block our access to any content that represents a threat to them, legitimate or otherwise.



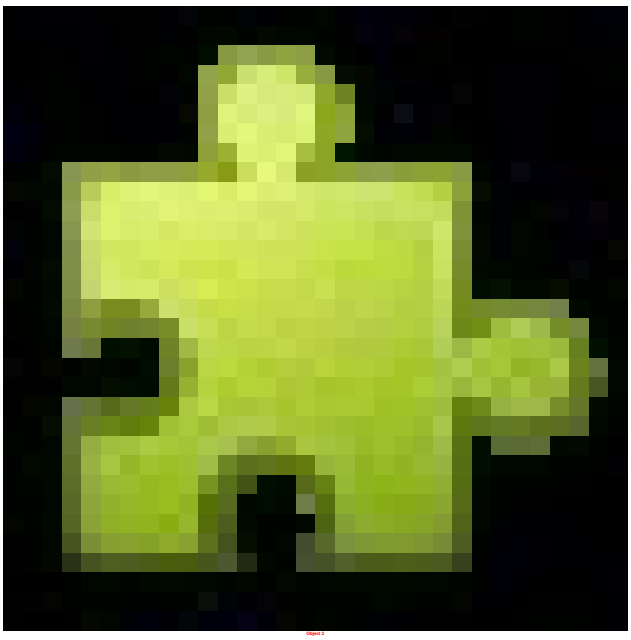
The web we were intended to have was **decentralized**, but the web we have today is very quickly becoming **centralized**, as billions of users become dependent on a small handful of services.

Regardless of whether you think this is a legitimate tradeoff, this was not how HTTP was intended to be used. Organizations like the NSA (and our future robot overlords) now only have to intercept our communications at a few sources to spy on us. It makes it easy for governments to censor content at their borders by blocking the ability for sites to access these highly centralized resources. It also puts our communications at risk of being interrupted by DDoS attacks.

Distributing the web would make it less malleable by a small handful of powerful organizations, and that improves both our freedom and our independence. It also reduces the risk of the "one giant shutdown" that takes a massive amount of data with it.

HTTP is inefficient

As of this writing, Gangnam Style now has over 2,344,327,696 views. Go ahead, watch it again. I'll wait for you.



above video url :: https://youtu.be/9bZkp7q19f0

Let's make some assumptions. The video clocks in at 117 Megabytes. That means (at most) 274,286,340,432 Megabytes, or 274.3 Petabytes of data for the video file alone has been sent since this was published. If we assume a total expense of 1 cent per gigabyte (this would include bandwidth and all of the server costs), $2,742,860 has been spent on distributing this one file so far.

That's not too bad... if you're Google. But if you're a smaller site, the cost to serve this much data would be astronomical, especially when bandwidth rates for small players start around $0.12 per gigabyte and go as high a $0.20 in Asia. I've spent the better part of my work at Neocities battling expensive bandwidth to ensure we can keep running our infrastructure at low cost.

HTTP lowered the price of publishing, but it still costs money, and these costs can really add up. Distributing this much data from central datacenters is potentially very expensive if not done at economies of scale.

What if, instead of always serving this content from datacenters, we could turn every computer on an ISP's network into a streaming CDN? With a video as popular as Gangnam Style, it could even be completely downloaded from within an ISP's network, not requiring numerous hops over the internet backbone. This is one of the many things IPFS is capable of improving (we'll discuss this in a bit).

HTTP creates overdependence on the Internet backbone

When content is hypercentralized, it makes us highly dependent on the internet backbones to the datacenters functioning. Aside from making it easy for governments to block and censor content, there are also reliability problems. Even with redundancies, major backbones sometimes get damaged, or routing tables go haywire, and the consequences can be drastic.

I got a weird taste of that a few months ago, when Neocities slowed down after a car crashed

into a fiber uplink we use in Canada (no suspects yet, but a few promising leads). I've also heard stories where hunters have shot at the fiber cables connecting the eastern Oregon datacenters (the enormous ones that store a *lot* of data), requiring engineers to show up on snowmobiles with cross country skis to repair the fiber lines. Since I wrote this post, details have emerged on a sophisticated attack on fiber lines happening in the Bay Area. The point is, the internet backbone isn't perfect, it's easy to attack it, and it's easy for service to get affected by a few important fiber lines getting cut.

****

Part 2: How IPFS solves these problems

****

We've discussed HTTP's problems (and the problems of hypercentralization). Now let's talk about how IPFS, and how it can help improve the web.

IPFS fundamentally changes the way we look for things, and this is it's key feature. With HTTP, you search for locations. With IPFS, you search for *content*.

Let me show you an example. This is a file on a server I run: https://neocities.org/img/neocitieslogo.svg. Your browser first finds the location (IP address) of the server, then asks my server for the file using the path name. With that design, only the owner (me) can determine that this is the file you're looking for, and you are forced to trust that I don't change it on you by moving the file, or shutting the server down.

Instead of looking for a centrally-controlled location and asking it what it thinks /img/neocitieslogo.svg is, what if we instead asked a distributed network of millions of computers not for the name of a file, but for *the content that is supposed to be in the file?*

This is precisely what IPFS does.

When neocitieslogo.svg is added to my IPFS node, it gets a new name: QmXGTaGWTT1uUtfSb2sBAvArMEVLK4rQEcQg5bv7wwdzwU. That name is actually a cryptographic hash, which has been computed from the contents of that file. That hash is guaranteed by cryptography to *always* only represent the contents of that file. If I change that file by even one bit, the hash will become something completely different.

When I ask the IPFS distributed network for that hash, it efficiently (20 hops for a network of 10,000,000) finds the nodes that have the data using a Distributed Hash Table, retrieves it, and verifies using the hash that it's the correct data. Early DHT designs had issues with Sybil attacks, but we have new ways to address them, and I'm very confident this is a solvable problem (unlike the problems with HTTP, which are just going to be broken forever).

IPFS is general purpose, and has little in the way of storage limitations. It can serve files that are large or small. It automatically breaks up larger files into smaller chunks, allowing IPFS nodes to download (or stream) files from not just one server like with HTTP, but hundreds of them simultaneously. The IPFS network becomes a finely-grained, trustless, distributed, easily federated Content Delivery Network (CDN). This is useful for pretty much everything involving data: images,

video streaming, distributed databases, entire operating systems, blockchains, backups of 8 inch floppy disks, and most important for us, **static web sites**.

IPFS files can also be special IPFS directory objects, which allow you to use human readable filenames (which transparently link to other IPFS hashes). You can load the directory's index.html by default, the same way a standard HTTP server does. Using directory objects, IPFS allows you to make static web sites exactly the same way you make them today. It's a single command to add your web site to an IPFS node: ipfs add -r yoursitedirectory. After that, it's available from any IPFS node without requiring you to link to any hashes in the HTML (example, and example with index.html renamed).

Federating data with IPFS

IPFS doesn't require every node to store all of the content that has ever been published to IPFS. Instead, you choose what data you want to help persist. Think of it like bookmarks, except instead of bookmarking a link to a site that will eventually fail, you back up the entire site for yourself, and volunteer to help to serve the content to others that want to look at it.

If a lot of nodes host a little bit, these little bits quickly add up to more space, bandwidth and availablity than any centralized HTTP service could ever possibly provide. The distributed web will quickly become the fastest, most available, and largest store of data on the planet earth. And nobody will have the power to "burn books" by turning it all off. This Library of Alexandria is *never* going to burn down.

Copying, storing and helping serve web sites from other IPFS nodes is easy. It just takes a single command and the hash of the site: ipfs pin add -r QmcKi2ae3uGb1kBg1yBpsuwoVqfmcByNdMiZ2pukxyLWD8. IPFS takes care of the rest.

IPNS

IPFS hashes represent immutable data, which means they cannot be changed without the hash being different. **This is a good thing** because it encourages data persistence, but we still need a way to find the latest IPFS hash representing your site. IPFS accomplishes this using a special feature called **IPNS**.

IPNS allows you to use a private key to sign a reference to the IPFS hash representing the latest version of your site using a public key hash (pubkeyhash for short). If you've used Bitcoin before, you're familiar with this - a Bitcoin address is also a pubkeyhash. With our Neocities IPFS node, I signed the image of Penelope (our site mascot) and you can load it using our IPNS pubkeyhash for that node: QmTodvhq9CUS9hH8rirt4YmihxJKZ5tYez8PtDmpWrVMKP.

IPNS isn't done yet, so if that link doesn't work, don't fret. Just know that I will be able to change what that pubkeyhash points to, but the pubkeyhash will always remain the same. When it's

done, it will solve the site updating problem.

Now we just need to make the location of these sites human-readable, and we've got all the pieces we need.

Human-readable mutable addressing

IPFS/IPNS hashes are big, ugly strings that aren't easy to memorize. So IPFS allows you to use the existing Domain Name System (DNS) to provide human-readable links to IPFS/IPNS content. It does this by allowing you to insert the hash into a TXT record on your nameserver (if you have a command line handy, run this: dig TXT ipfs.git.sexy). You can see this in action by visiting http://ipfs.io/ipns/ipfs.git.sexy/.

Going forward, IPFS has plans to also support Namecoin, which could theoretically be used to create a completely decentralized, distributed web that has no requirements for a central authority in the entire chain. No ICANN, no central servers, no politics, no expensive certificate "authorities", and no choke points. It sounds crazy. It is crazy. And yet, it's completely possible with today's technology!

IPFS HTTP gateway: The bridge between the old web and the new

The IPFS implementation ships with an HTTP gateway I've been using to show examples, allowing current web browsers to access IPFS until the browsers implement IPFS directly (too early? I don't care). With the IPFS HTTP gateway (and a little nginx shoe polish), we don't have to wait. We can soon start switching over to IPFS for storing, distributing, and serving web sites.

How we're using IPFS now

Our initial implementation of IPFS is experimental and modest, for now. Neocities will be publishing an IPFS hash once per day when sites are updated, accessible from every site profile. This hash will point to the latest version of the site, and be accessible via our IPFS HTTP gateway. Because the IPFS hash changes for each update, this also enables us to provide an archive history for all the sites, something we automagically just get from the way that IPFS works anyways.

How we'll use IPNS in the future

Long-term, if things go well, we want to use IPFS for storing all of our sites, and issue IPNS keys for each site. This would enable users to publish content to their site independently of us. If we do it right, **even if Neocities doesn't exist anymore, our users can still update their sites**. We effectively take our user's central dependence on our servers and smash it to pieces, permanently ruining our plans for centralized world domination forever. It sounds awesome. It is awesome!

It's still early, and there's much work to do before IPFS can replace HTTP without needing to describe the idea as crazy. But there's no time like the present to plan for the future. It's time for us to get to work. Accept the Internet Archive's challenge: **distribute the web**.

Link ::

https://ipfs.io/ipfs/QmNhFJjGcMPqpuYfxL62VVB9528NXqDNMFXiqN5bgFYiZ1/its-time-for-the-permanent-web.html

****

Ipfs installayion and configuration

****

download the go-ipfs from the link
https://ipfs.io/docs/getting-started/

After downloading, untar the archive, and move the ipfs binary somewhere in your executables $PATH:

 # tar xvfz name.tar.gz
 # mv go-ipfs/ipfs /usr/local/bin/ipfs

Test it out:

 # ipfs help

init the repo

 # ipfs init

initializing ipfs node at /Users/jbenet/.go-ipfs
generating 2048-bit RSA keypair...done
peer identity: Qmcpo2iLBikrdf1d6QU6vXuNb6P7hwrbNPW9kLAH8eG67z
to get started, enter:

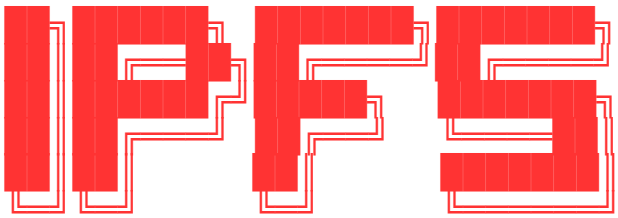
ipfs cat /ipfs/QmYwAPJzv5CZsnA625s3Xf2nemtYgPpHdWEz79ojWnPbdG/readme

Note the hash there may differ. If it does, use the one you got.
Now, try running:

 # ipfs cat /ipfs/QmYwAPJzv5CZsnA625s3Xf2nemtYgPpHdWEz79ojWnPbdG/readme

You should see something like this:

Hello and Welcome to IPFS!



If you're seeing this, you have successfully installed
IPFS and are now interfacing with the ipfs merkledag!

```
 ----------------------------------------------------
| Warning:                                           |
|   This is alpha software. use at your own discretion! |
|   Much is missing or lacking polish. There are bugs.  |
|   Not yet secure. Read the security notes for more.   |
 ----------------------------------------------------
```

Check out some of the other files in this directory:

```
  ./about
  ./help
  ./quick-start     <-- usage examples
  ./readme          <-- this file
  ./security-notes
```

You can explore other objects in there. In particular, check out quick-start:

 # ipfs cat /ipfs/QmYwAPJzv5CZsnA625s3Xf2nemtYgPpHdWEz79ojWnPbdG/quick-start

Going Online

Once you're ready to take things online, run the daemon in another terminal:

 # ipfs daemon

Initializing daemon...
API server listening on /ip4/127.0.0.1/tcp/5001

Gateway server listening on /ip4/127.0.0.1/tcp/8080

Wait for all three lines to appear.

Make note of the tcp ports you get. if they are different, use yours in the commands below.

Now, if you're connected to the network, you should be able to see the ipfs addresses of your peers:

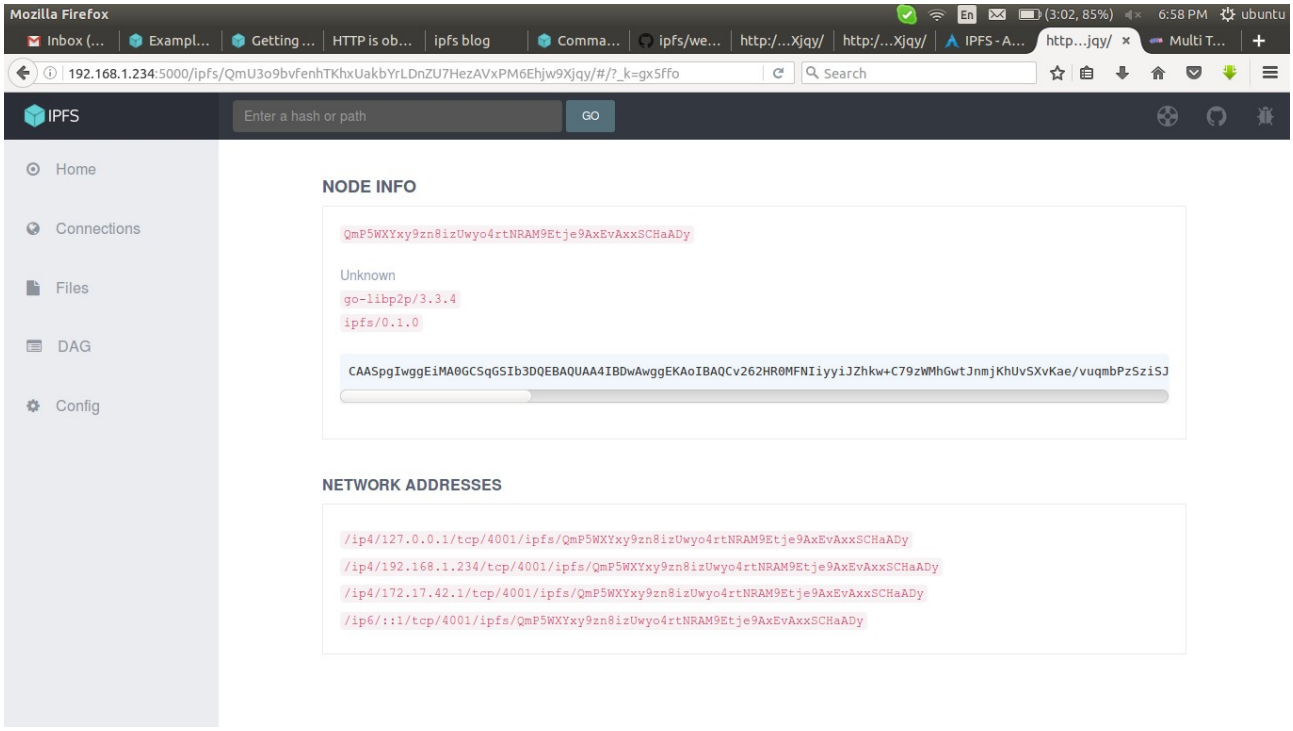
 # ipfs swarm peers
/ip4/104.131.131.82/tcp/4001/ipfs/QmaCpDMGvV2BGHeYERUEnRQAwe3N8SzbUtfsmvsqQLuvuJ
/ip4/104.236.151.122/tcp/4001/ipfs/QmSoLju6m7xTh3DuokvT3886QRYqxAzb1kShaanJgW36yx
/ip4/134.121.64.93/tcp/1035/ipfs/QmWHyrPWQnsz1wxHR219ooJDYTvxJPyZuDUPSDpdsAovN5
/ip4/178.62.8.190/tcp/4002/ipfs/QmdXzZ25cyzSF99csCQmmPZ1NTbWTe8qtKFaZKpZQPdTFB

Fancy Web Console

We also have a web console you can use to check the state of your node. On your favorite web browser, go to:

 # http://localhost:5001/webui

This should bring up a console like this:



to change the access address from localhost to ip or any other ::

here i use 192.168.1.234 as my peer and port 5001

 # nano /{path to .ipfs}/config

change

```
  "Addresses": {
    "API": "/ip4/127.0.0.1/tcp/5000",
    "Gateway": "/ip4/127.0.0.1/tcp/8080",
    "Swarm": [
      "/ip4/0.0.0.0/tcp/4001",
      "/ip6/::/tcp/4001"
```

to

```
  "Addresses": {
    "API": "/ip4/192.168.1.234/tcp/5001",
    "Gateway": "/ip4/192.168.1.234/tcp/8080",
    "Swarm": [
      "/ip4/0.0.0.0/tcp/4001",
      "/ip6/::/tcp/4001"
```

 # ipfs config --json API.HTTPHeaders.Access-Control-Allow-Origin '["http://192.168.1.234:5001"]'
 # ipfs config --json API.HTTPHeaders.Access-Control-Allow-Methods '["PUT", "GET", "POST"]'
 # ipfs config --json API.HTTPHeaders.Access-Control-Allow-Credentials '["true"]'

stop the daemon and start it again,

 # ipfs daemon

Initializing daemon...
Swarm listening on /ip4/127.0.0.1/tcp/4001
Swarm listening on /ip4/172.17.42.1/tcp/4001
Swarm listening on /ip4/192.168.1.234/tcp/4001
Swarm listening on /ip6/::1/tcp/4001
API server listening on /ip4/192.168.1.234/tcp/5000
Gateway (readonly) server listening on /ip4/192.168.1.234/tcp/8080
Daemon is ready

now it is accesible at

http://192.168.1.234:5001/webui

There are public gateways, allowing users with no IPFS node running to access files on the network.

http://ipfs.io/ipfs/QmYwAPJzv5CZsnA625s3Xf2nemtYgPpHdWEz79ojWnPbdG

where, QmYwAPJzv5CZsnA625s3Xf2nemtYgPpHdWEz79ojWnPbdG is the hash of file uploaded.

****

Links ::

full knowledge >>
https://ipfs.io/docs

installation >>
https://ipfs.io/docs/getting-started/

basic commands >>
https://ipfs.io/docs/commands/#ipfs

access webui with different domainname than localhost >>
https://github.com/ipfs/webui

public gateway and services >>
https://wiki.archlinux.org/index.php/IPFS

ipfs in docker >>
https://ipfs.io/blog/1-run-ipfs-on-docker/

some commands and usage >>
https://ipfs.io/docs/examples/

****