IPFS Pinning UX PRD

Summary

The IPFS abstraction called “pinning” serves an important purpose at the protocol level (“do not GC this block or DAG!”), but does not, at a human-conceptual level, do what end users or developers expect. **Users/developers expect to interact with IPFS by “adding” or “saving” or “putting” objects there, without reference to “pinning.” They also expect their objects to be quickly available on IPFS after “adding,” and to persist on IPFS until they take some other action. Right now, none of these expectations are being met.** More advanced dweb concepts (such as collaborative data stewardship, or co-hosting, that significantly reimagine the end user network experience) are even less well-understood, in part because these basic “adding” concepts aren’t yet clear.

These issues should be addressed in two phases:

- **One, we need to get pinning abstractions and language right.** Currently, our semantics are muddled in being a network and a file system at the same time. We should start clarifying

the pinning user journey from the perspective of a file system, and grow towards the dweb network concepts, as we should not expect users to depart Web 2.0 concepts in one fell swoop. For example, internally when we talk about "saving" on IPFS, our journeys often start with "I saw this thing on IPFS and want to save it," when in reality, most people's journeys start with "I have a file on my local machine that I want to put on IPFS," ie, **for most people, saving is an 'upwards' motion 'to' to network, rather than a 'downwards' motion 'from' the network** and we should design our initial user workflows to match this expectation, adding in more complex concepts around the networking nature of IPFS as a separate iteration -->

- **Two, we need to use our interfaces to teach users about data availability and persistence on the dweb.** [MOAR]

Background / current state / pain points

What is a pin?

In IPFS, a "pin" is a DAG (for recursive-pins, as identified by its root CID) or a block (for direct-pins), that is not to be removed during Garbage Collection (GC). This "saves" the pinned information so that it is available for the user in the future, or available for others to discover and perhaps view or "save/pin" themselves. The pinned information is discoverable to the extent that the peer itself is accessible on the network (ie, if pinned to an IPFS network running privately, then it is only available within that private network; by default, an object pinned on a single peer is available on the public IPFS network).

From a semantics perspective, and to paraphrase @whyrusleeping from this note, IPFS tries to make it feel as though all objects are local. There is no "retrieve this file for me from a remote server;" the commands all act the same way no matter where an object is located. However, users want to be able to control what they keep. Pinning is the mechanism that allows a user to tell IPFS to always keep a given object local and accessible. IPFS caches objects locally for a short time after the user performs any IPFS operation on them, but these objects may get garbage collected eventually. To prevent this, users can "pin" the hash they care about. Objects added through `ipfs add` are pinned recursively by default because IPFS assumes that if you go to the trouble of adding an object to the network, you care about it enough to want to keep it around.

Types of pinning

Pinning process

- Sync: the user needs to wait until it is pinned (ipfs).

    - 😕Users must wait until sync pinning is 100% complete until they can do another action or end a process, and the objects being pinned must continue to be accessible throughout (aka, don't close your laptop)
- Async: the user submits the pin (Cluster, pinning services) and the service pins in the background
    - 🎉Intermediary services are providing a layer of tooling that makes it easier for people to engage with "saving" objects on the dweb.
    - 😕Even for these intermediary services, communicating that it takes some amount of time between when you "save" or "add" something to IPFS, and when it's actually available on IPFS, is a challenge.

Pin types

- Recursive (pins a given block and all of its children)
    - This is the type of pinning most end users and developers are concerned with.
- Direct (pins a single block)
- Indirect (what you call a block that's been pinned recursively; a child of a recursively-pinned block)

Types of pinning interactions

Pinning to a single peer (ipfs)

- Commands (CLI)
    - `ipfs add`
        - "I have an object on my machine, and I want to put it on IPFS." This is the idea that most end users and developers associate with adding.
        - Can only take raw bytes.
        - 🎉By default, this command pins after adding. IPFS assumes that if a user goes to the trouble of adding an object to IPFS, they also want to keep it around. This meets end user and developer expectations.
    - `ipfs pin add`
        - Basic: "I already have an object in my datastore, and I want to keep it around by "saving" it."
            - Someone sends you a quokka picture over IPFS.
                - "Hey I think you will like this picture"
                - "Sure let me have a look" (you look at the photo)
            - That's a great quokka, and you want to keep it so that it doesn't get GC'd/cleared from cache/cleared from datastore. It's in your datastore already because you viewed it over IPFS.
                - `ipfs pin add`

        - Advanced dweb: “I already have an object in my datastore, and I want to help make it available on the network by co-hosting it.”
            - You are browsing Wikipedia via IPFS, and you want to make sure some very important quokka information is always available.
            - Since you are browsing via Companion with a local IPFS node, once you’ve viewed that Wikipedia page, its objects are already in your datastore. You want to keep them so they don’t get GC’d, and are available via your node.
                - `ipfs pin add`
        - Can only take a hash.
    - `ipfs add` vs `ipfs pin add`
        - `ipfs add` will take a ‘raw’ file and chunk and add it to the datastore, whereas `ipfs pin add` needs the file to exist in the network, and will actually ask the network for it if it doesn’t already exist in the local datastore.
        - 😕These commands are confusing in part because although they both use the word “add,” these are not the same “add.” Ideally, these actions would have been called something more clearly different in our API.
            - `ipfs add` = chunk my file into my datastore/network (`add`), and don’t delete them (the `pin` that happens automatically after `add`)
            - `ipfs pin add` = don’t GC these existing datastore/network blocks (`pin` -- and what’s that `add` for? Shrug. Certainly not file chunking, as in `ipfs add`.)
        - 🎉Fortunately, we can use different terms for end users, and hide this complexity. If we could, it would be even *better* to rename `ipfs pin add` to just `ipfs pin` or another solution that doesn’t involve the word `add`.
    - `ipfs pin rm`
        - IPFS does not remove blocks. It justs removes a given <cid> from the list of recursive pins.
        - In order to remove the unpinned blocks from IPFS (so that they will not be advertised or provided to the network), IPFS garbage collection (`ipfs repo gc`) needs to run. This checks the list of pins, and traverses recursively for every pin. It puts all the blocks that are indirectly referenced from the root pins in a bag, and then removes blocks that are *not* in that bag.
        - 😕This is a very expensive procedure, and while this happens nothing else can happen; the peer becomes unusable.
        - 😕There isn’t a way for a node to express “I'm not providing that anymore” on the DHT. Records expire after 12 hours, but even if the peer has GC'ed it may still be contacted for blocks it doesn't have until that happens.
- APIs

    - HTTP API: POST /v0/api/pin/add, /v0/api/add
    - API Bindings: `Pin/Add` methods in Go and JS that trigger HTTP API requests
    - Core API: Pin method can be used to pin when programmatically running an IPFS node

Single peer + GUI

- IPFS WebUI
    - IPFS WebUI is a browser-based interface for a user's IPFS node. Via this interface, users can check on node stats, explore the IPLD-powered merkle forest, see peers around the world, and manage their files, without needing to touch the CLI.
    - It is a React app that communicates with a user's local node via ipfs-http-client.
    - Regarding pinning:
        - WebUI is both a view into what is pinned and what files/folders you have in your MFS (your local datastore in general can have blocks that are not displayed on the Files screen), and a place where you can take actions and have those reflected across IPFS.
        - 🎉You can graphically "+Add"
            - File
            - Folder
            - From IPFS
            - New Folder
        - Once a file or folder is in your Files interface, you can:
            - View the file or folder.
            - Delete
                - 😕See `ipfs pin rm` above for UX issues with 'delete'
            - Rename
                - 🎉Works as advertised.
            - Download
                - 🎉Works as advertised.
            - Inspect
                - 😕Opens file or folder in "Inspect" page. It is not very clear what this does, or what it's for, when coming from the Files page.
            - Copy hash
                - 😕Works as expected and copies QmHash to clipboard, but there's no UI indicator that it's been successful, as with other interactions. This makes the user unsure if the action has 'worked.'
            - Share

        - 😕Opens a modal for a user to copy a hash that's prepended with an http path to the item. This does not meet user expectations because Web 2.0 "Share" icons typically provide several direct-share-to-app options plus something called "copy link" or similar. The WebUI "Share" is actually == "copy link," and users are being asked to "link" to something that isn't usually linked to in today's Share systems (ie, one would typically "send" or "share" a doc or image directly with someone, not send them a link to the item; links are for webpages and similar). There is an opportunity to improve this interaction quite a lot, as even in file systems such as Dropbox, linking and sharing isn't always straightforward.
    - Pin or Unpin
        - 😕It is not clear how the 'pin' icon and options on the "files" part of the Files Page relate to the pins on the "pins" part of the Files Page. The numbers of items differ. Also, the files in "files" are not GC'd, so why don't they *all* have pin icons? What does the pin icon mean here?
    - 😕There are different options depending on whether the user selects the drop-down, or selects the checkbox next to the item.
- In the Pins interface, you can do a subset of the actions above on the pin directly:
    - Share
    - Copy hash
    - Download
    - Inspect
    - Unpin
    - 😕These have the same positive/negative notes as noted above in 'files', with the additional confusion of -- how can I, as a user, decide which pin I want to act on? All that's listed are hashes with no other identifying information, and in fact, those hashes are listed twice (as item title, and in the slot where the hash would go if there *were* a proper title for that item).
- 😕There's a confusing dichotomy between the "files" and "pins" on the File Page. Right now, "**files**" in WebUI shows only files and directories in MFS, and MFS is a subset of everything in local datastore:
    - `ipfs add` adds data to the local local datastore (repo), but it does not add it to MFS. To add a file or directory to MFS via CLI, you need to:
        - `ipfs add` to get CID

        - Add the CID to MFS via `ipfs files cp /ipfs/{CID} /name-on-mfs`
      - When the user adds a file or a directory via WebUI, it does both steps (`ipfs add` + `ipfs files cp` to MFS)
  - ...while "**pins**" shows objects that the user has "pinned" (`ipfs pin add`), even though both are technically "pins" behind the scenes, and the user expects to see both via the same interface (ie, I want to see what I've got in my local repo).
    - 😕Confusion points for users: Why can I see the names of files, but not names of pins? Why can I take certain actions on files/folders, but not on pins? Why, when I inspect files versus pins, or when I inspect different files, do I see different things? Some have more file-looking items there with names, some have one big dot, some have lists of hashes without real titles associated with them. There's a confusing lack of intention around what is shown as a "file," a "pin," and what the interface on the "Inspect" page means (which of what is shown is a "file" or a "pin" or something else?).
  - 🎉The GUI team is aware of many of these painpoints, and has existing plans to improve this interface that will be informed by this PRD
- IPFS Desktop
  - IPFS Desktop allows users to run an IPFS node on their machines without having to bother with command line tools. With it, they have the power of the IPFS WebUI, plus a handful of helpful shortcuts. It is available on Mac, Windows, and Linux.
  - IPFS Desktop's main feature is to allow you to have the IPFS daemon always running in the background, and it provides auto-update mechanism, which makes sure user is running the latest version of IPFS daemon behind the scenes.
  - Regarding pinning:
    - On Windows, you can right click on files and folders to add them to IPFS.
    - On macOS, you can drag and drop objects to the tray icon to add them to IPFS.
    - To view added files, the user has the same experience as via the WebUI (see above).
- IPFS Companion
  - IPFS Companion is a browser extension that enables everyone to access IPFS objects the way they were meant to be accessed: from a locally-running IPFS node. It is available on Firefox, Firefox for Android, Chrome, Brave, Opera, and Edge.
  - Companion detects requests on websites for IPFS-like paths (/ipfs/{cid} or /ipns/{peerid_or_host-with-dnslink}), and redirects and loads them from a local gateway:

        - https://ipfs.io/ipfs/QmbWqxBEKC3P8tqsKc98xmWNzrzDtRLMiMPL8wBuTGsMnR → http://127.0.0.1:8080/ipfs/QmbWqxBEKC3P8tqsKc98xmWNzrzDtRLMiMPL8wBuTGsMnR
    - Companion will also detect the presence of DNSLink in DNS records of visited websites and redirect HTTP requests to a local gateway (the browser will load the website from IPFS).
        - http://docs.ipfs.io → http://127.0.0.1:8080/ipns/docs.ipfs.io
    - Companion provides additional actions for pages loaded from IPFS:
        - Pin/Unpin of IPFS resources (via API)
        - Copy canonical IPFS address
        - Copy shareable URL to resource at preferred public gateway
    - Regarding pinning:
        - Has "Share files via IPFS" in its browser extension pop-up menu. Clicking on this opens a new window where you can upload a file or directory. Doing so adds that item to IPFS (and pins by default unless a checkbox is unchecked).
            - 🎉There is a plan to change this behavior and add files to MFS instead, so the user can see shared files in WebUI's Files screen.
        - Adds the ability to right click, and “Add selected text to IPFS.” This opens the selected text in a new browser window with its QmHash+http-accessible-url in the address bar. This new pin is then listed in the the user’s WebUI/Desktop “Files Page/pins”.
            - 😕What is the user need behind selecting text and adding it to IPFS?
- IPFS GUI apps <> public IPFS network <> expectations of privacy
    - Our GUI apps, by default, only interact with the public IPFS network. All pins and files displayed via their interfaces are technically findable by other IPFS nodes. (One can point an IPFS node to a private network, if desired, and Companion is just a proxy to IPFS nodes.)
    - 😕In a distributed system in which other people can “save”/pin information, sometimes data is neither available in the way folks have gotten used to on Web 2.0, and nor is it “deleted” in the way folks understand that concept today.
    - 😕There is a mismatch between technical network privacy and what end users expect “local” and “privacy” to mean. This is a UX hurdle that should be treated with care as we move forward, as people expect their local machine to automatically be “private” in the way they understand “this is my house versus that is your house, and I need to invite you in before you can see what's inside,” unless this is clearly messaged otherwise (ex dropbox / google drive folders).

Pinning to multiple peers (Cluster)

- Commands (CLI)
    - `ipfs-cluster-ctl add`
        - 🎉By default, this command pins after adding.
        - The `ipfs-cluster-ctl add` command is very similar to `ipfs add` except that the `ipfs add` command only adds to a local IPFS peer, and `ipfs-cluster-ctl add` adds to several Cluster peers at the same time. How many it adds to depends on the replication factors the user sets as command flags or the defaults in the configuration file.
        - 🎉Cluster considers a `pin add` operation to be successful when the cluster-pinning stage is finished. This means the pin has been ingested by Cluster and that things are underway to tell IPFS to pin the content. If IPFS fails to pin the content, Cluster will know, report about it and try to handle the situation. It works this way because cluster-pinning stage is relatively fast but the ipfs-pinning stage can take days. Therefore, the second stage happens asynchronously once the cluster-pinning stage is completed.
    - `ipfs-cluster-ctl pin add`
        - Adds content from the IPFS network to a Cluster. The `ipfs-cluster-ctl pin add` operation is similar to the `ipfs pin add` one, but allows the user to set Cluster-specific flags, such replication factors or the name associated with a pin.
    - `ipfs-cluster-ctl pin rm`
        - Pins can be removed from a Cluster at any time. They are then treated by IPFS as any other `ipfs pin rm` command.
- Cluster APIs
    - HTTP
        - Rest API: POST /pins/
        - Proxy API: POST /v0/api/pin/add, /v0/api/add
    - API Bindings: `Pin/Add` methods in Go and JS that trigger Rest API requests
    - Go API: Pin method can be used to pin when programmatically running an IPFS Cluster peer
- Types of Clusters
    - Standard Cluster: all peers have equal “status;” they all control pinning/unpinning. All data on a Standard Cluster replicates across all peers.
        - 🎉Cluster provides a layer of tooling that makes IPFS / availability on the dweb easier to understand and develop on because it solves for a very common need (automated availability) that developers or end users would, first, have to understand is *missing* from single-peer IPFS usage (no small feat for those new to the space), and second, otherwise have to build themselves.

    - Collaborative Cluster: some peers have less "status;" they are followers that can contribute storage, but cannot control what gets pinned/unpinned. Full peers in a Collaborative Cluster control what gets pinned/unpinned, and their decisions replicate across all peers.
        - 🎉Users have requested this feature from IPFS, and we already have it!
        - 😕But they're requesting it because they don't know that we already have it.

Third-party pinning services

- Pinning services (such as Pinata, Infura, and 3box) offer APIs and browser interactions that act like Amazon S3 and similar Web 2.0 data store services.
    - 🎉Services like these allow developers to build on the dweb without requiring them to replace their entire stack/mental model of development in one go.

Goals / definition of success

We believe that:

- Defining how users should represent, interact with, and reason about their pins in IPFS across IPFS entry points

Will:

- Make it easier for end users and developers to understand how to use IPFS as a file system, and make data available and persistent on the dweb

We will know that we are right when:

- The number of users creating apps and participating in the IPFS ecosystem using the "happy paths" we provide increases.
- 1️⃣IPFS-as-file-system (IPFS Desktop/WebUI)
    - The number of repeat users of IPFS Desktop/WebUI increases
    - The number of objects and/or amount of data stored via IPFS Desktop/WebUI increases, globally.
    - Testing shows that users can accurately explain how to add, share, and delete information from the dweb (and any related limitations on those actions)
- 2️⃣Data availability (IPFS Cluster/Collaborative Clusters)
    - The number of Collaborative Clusters / Clusters that have follower peers increases
    - The number of repeat users of/interactions with Collaborative Cluster increases
    - Testing shows that users understand what happens when you have objects on one IPFS node and that node goes offline.
    - Testing shows that users can accurately explain how to make their information available on the dweb (and any related limitations).
- 3️⃣Data persistence (Co-hosting/Pinning Services)

- The percentage of co-hosted information on IPFS versus single-node information increases. (Co-hosted in this case can mean self-co-hosted, or third-party co-hosting via a pinning service, but the idea in general is -- the object is pinned to more than one node.)
- Testing shows that users understand what guarantees IPFS alone provides re: persistence, and the added value of co-hosting information or using a pinning service.

User workflows

1️⃣ IPFS-as-file-system: John Q. Curious Public wants to stay in control of his data, and share pictures with his family.

North star

A non-developer end user should not need to know what 'pinning' is to get the benefits of IPFS. We should abstract the idea of 'pinning' away by re-branding MFS as "IPFS Drive" and build our GUI products as different interfaces into one user experience.

Priorities

In the short term, the low hanging fruit is to move our GUI applications from using the low-level Pin API to adding user data to MFS, where all files are implicitly pinned, and are also much easier to manage via Web UI. We should also transition the language in our GUI apps (both visual and words) away from 'pinning' and towards clear, non-technical vocabulary.

Note: we don't have any official presence on mobile yet, so any mobile app (iOS or Android) would a new endeavour. This scope of work assumes, for now, focusing on improving our existing IPFS Desktop and browser-based applications.

Workflow

"I want to share pictures with my family, but I don't want to lose control of where my pictures are. Those computer companies can see everything! Who knows what they're using my stuff for."

User workflow	**IPFS interaction**
John Q. **downloads and installs** the IPFS Drive app (from the location that works for his computer)	On installation, IPFS Drive sets up common default folders in a pre-defined location on the user's machine. These include Public and Private folders. [^1]
John Q. **drags and drops** pictures into his new IPFS Drive/Private folder.	Everything in IPFS Drive (née MFS) is implicitly pinned (ie, it won't be garbage-collected unless

	explicitly removed from MFS). The pictures/data in the Private folder are *not* announced to the network.
John Q. wants to **share** a specific photo with his daughter. He opens his email in his browser, and drags and drops a file from his IPFS Drive into a new **email**, like he always does.	When a file is attached to something outside of the IPFS Drive/Private folder, the equivalent of "share with anyone who has this link" is created for access to that file. [<-- this needs more feedback from the team]
John Q.'s daughter gets an email from her dad, and can **see** the image he sent in email preview, and can **download** by clicking on the link.	IPFS Drive allows you to share a permalink to every file or directory, which can be accessed via http for those who do not use IPFS.
As John Q. learns more about IPFS, he wants to help **host** information. He saves or moves the things he wants to help host to his IPFS Drive/Public folder.	The pictures/data in the Public folder are announced to the network. This is where participation in the global IPFS network happens.
John Q. **deletes** an accidental copy of a photo in his Private folder by dragging and dropping the photo into his computer's trash.	When the photo is moved out of IPFS Drive/Private and into Trash, IPFS Drive deletes all local blocks that the deleted object linked to, as long as they're not relied on by another block.
John Q. is running out of space on his machine, and decides to **stop hosting** some photos he downloaded from Wikipedia by dragging and dropping them from his Public folder into his computer's trash.	When the photo is moved out of IPFS Drive/Public and into Trash, IPFS Drive deletes all local blocks that the deleted object linked to, as long as they're not relied on by another block.

Notes

- [^1] In a distributed system in which other people can save/pin information, sometimes data is neither available in the way folks have gotten used to, and nor is it 'deleted' in the way folks understand that concept today, and so the collection of 'IPFS Drive' interfaces need to treat these interactions with care and intention.

  Suggestion: intentional implementation of Public v Private folders in IPFS Drive, where Public means if you put something in here, expect it to be on the internet forever (like the burrito pic I put on Yelp), and Private means that you have strong, nuanced controls over who sees and interacts with what.
- The relationship between pinning and MFS: https://gist.github.com/meiqimichelle/1e4601b4418bf4f46007f4777aff395d

2 Data availability: Data Architect Maria needs to replicate huge datasets across many data centers to make it highly available for scientific study.

North star

Someone managing large data (either in number of files or in volume) should not need to copy-and-paste hashes or peer ids into the standard IPFS CLI to do their work. We need to actively market and develop IPFS Cluster as the entry point for the 'enterprise management' use case. We also need to build a lightweight admin interface that makes it easy for data administrators to manage pins and peers.

Priorities

In the short term, we should re-write the IPFS and IPFS Cluster homepages (and any other useful communication points) to actively market IPFS Cluster as our 'enterprise admin' solution, and provide examples of how to set up 'Collaborative Clusters' with 'follower' peers. We should also build a very simple, v0 enterprise admin panel that controls and explains several key IPFS data admin concepts (for example: replication factors; types of Clusters; pinsets; and followers).

Workflow

"As a data architect/IT lead, I need to be able to manage and move massive datasets across my data and research centers to support my scientists."

User workflow	**IPFS interaction**
Maria **download and installs** IPFS Cluster on a local data center server. It suggests spinning up three IPFS peers in her Cluster as a start, and provides tips around what she should expect when adding objects of the type she has (ie, it can be slow when adding large datasets to her peers for the first time; what sort of network settings she should have; etc).	As part of the installation flow, IPFS Cluster asks a few short questions about data size and shape, and then spins up a Cluster with settings that are most likely to meet that users' starting needs. It also prompts the user to visit the admin panel.
As prompted, Maria **explores** her new IPFS Cluster **admin panel**, and learns how she can change replication factors; configure various types of Clusters; set permissions across peers; manage her pinsets; and more.	The Cluster admin panel is functional, but also educational, especially when someone installs Cluster for the first time, as people need to know *why* certain choices might be made across its options.
Via the admin panel, Maria **creates a 'followers'** link so that she can ask scientists across her data centers to join her Cluster. Copying-and-pasting her new link also copies convenient instructions on how to install	The admin panel provides an easy way to add follower peers to a given Cluster. These are peers that participate in hosting information in a Cluster, but don't have equal ability to impact Cluster settings in other ways.

Cluster locally that she can edit to meet her specific needs (ie, her scientists all work on high-performance machines because of the nature of what they do, and so she can tailor the instructions for her team's needs, ad-libs style).	
As scientists click her followers link and follow the instructions, she **sees their peers join her Cluster** via the admin panel. She can **explore performance metrics** across her Cluster, and troubleshoot issues via logs.	The admin panel gives data managers visibility into their system of peers, and either provides performance metrics and logs there, or links to other IPFS tools/views that provide that information.
Maria **programmatically moves** large datasets to IPFS via the Cluster CLI.	Cluster replicates the information the user adds, and adds+pins it automatically to IPFS. The user can see metadata about their collection of pins (pinsets) via the admin panel.
[sharding]	[sharding]
When scientists leave her organization, Maria can **remove** their peers as **followers** via the admin panel.	The Cluster admin panel allows the admin to add and remove peers as needed. When this happens, Cluster automatically rebalances data load across the remaining peers.

Notes

- IPFS Cluster <> Filecoin Integration: https://docs.google.com/document/d/1BUt7stI6gtIBuLrQYzNJ5hEdRaHrbU3a-Um7MfyYAYw/edit#
- Cluster Summit trip report, see "collaborative cluster" and "federated cluster": https://github.com/protocol/event-management/issues/147#issuecomment-480694315v
- Enterprise package management use case (ipfs-cluster): https://github.com/ipfs/user-research/issues/1

3 Data persistence: John Q. Curious Public wants his photos to persist on the distributed web.

North star

People need support for persisting data on IPFS, either via clearer paths to co-hosting, or via third-party pinning services. We need to convey how 'saving' something on IPFS, or more importantly, 'publishing' or 'sharing' something on IPFS, doesn't necessarily mean that it's always accessible.

Priorities

In the short term, we should provide clear indicators (both visual and words) of what saving, publishing, and sharing, mean in the IPFS ecosystem. We should also give people next steps in our GUI applications for data persistence, which today may be as simple as explanations and links pinning services and roll-your-own co-hosting information.

At a deeper technical level, we need to improve our garbage collection user experience. As traversing a DAG to perform GC is essentially unavoidable, we should intentionally experiment with when and how this happens, and how many other things can happen concurrently. Also, we need to address the race condition problem in the API. [^2]

3.1 Workflow: pinning service

Hacker "Pinning Service" MacDev: "I really believe in the distributed web, and I want people to have a pleasant user experience around making their data persist and be available."

John Q.: "I want my photos to stick around on IPFS, but I'm out of space on my personal computer, and I don't want to, like, buy more computers or learn how to run a server or whatever it is I need to do."

Customer workflow	**Intermediary workflow**	**IPFS interaction**
John Q. **signs up for** an account on a **pinning service**.	Hacker MacDev's pinning service manages user accounts.	No immediate IPFS interaction.
John Q. clicks "upload" and **browses via the pinning service** for the directory called "Q. Curious Family History," and **selects** it.	Hacker's service allows browsing for and uploading local directories to IPFS. It does this by tracking IPFS hashes as well as its own metadata.	The pinning service "adds and pins to IPFS" everything added to its interface.
John Q. sees a notice estimating the amount of time it will take for his directory to be available on IPFS.	Hacker's service **adds** John Q.'s directory to IPFS in the background.	IPFS recursively pins John Q.'s directory.
John Q. gets a notification when his directory is available on IPFS.	Hacker's service **replicates** John Q.'s directory across a Cluster of peers so that his information is highly available.	IPFS Cluster is used to orchestrate data replication.
John Q. **pays** for the amount of storage he uses.	Hacker's service tracks the amount of storage each user has on IPFS, and charges them accordingly. [^3]	The IPFS Pinning Service API provides the information the pinning service needs to track the size and number of pins.
John Q. **updates his directory**	Hacker's service interfaces with	IPFS pins new files in the

(adding and deleting some photos, changing names of others), and **re-uploads** it to his pinning service.	IPFS to add, delete, and rename files. The service replicates data such that there's **no downtime** while IPFS unpins and runs GC. [^4]	directory, removes pins and blocks that should no longer exist, runs GC, and creates a new hash for the root directory.

3.2 Workflow: community co-hosting

Librarian Alex: "My library is a trusted home for my community. Now that so much of life is digital, I want it to be a home for my community's digital stories and histories, too."

John Q.: "I'm getting the hang of this 'data stewardship' thing, and want to host my family history. The pinning service is nice, but that's a for-profit enterprise. Maybe there's another option."

Customer workflow	**Intermediary workflow**	**IPFS interaction**
John Q. claims his account on his local library's Cube.	Librarian Alex's Cube **manages membership accounts** via library card number. Each member gets 10 GB of storage on the Community Cube, which Alex hosts on AWS.	No immediate IPFS interaction.
John Q. clicks "upload" and **browses via the Cube interface** for the directory called "Q. Curious Family History," and **selects** it.	The Cube online interface allows browsing for and uploading local directories to IPFS, up to limits set by an administrator. It does this by tracking IPFS hashes as well as its own member metadata.	Cube "adds and pins to IPFS" everything added to its interface.
John Q. sees a notice estimating the amount of time it will take for his directory to be available on the Community Cube.	Cube **adds** John Q.'s directory to IPFS in the background.	IPFS recursively pins John Q.'s directory.
John Q. gets a notification when his directory is available on the Community Cube.	Alex's Cube **replicates** John Q.'s directory across a Cluster of peers so that his information is highly available.	IPFS Cluster is used to orchestrate data replication.
John Q. **sends a link** to the Community Cube. His	Cube **provides a central interface on the IPFS**	The IPFS Pinning Service API provides the information Cube

daughter can see his information, as well as other directories that other members have contributed.	**Gateway** that shows all 'contributed' information. The Cube provides further availability for this information, in addition to members' local Public IPFS folders. [Could this be self-hosted/ self-gateway'd?]	needs to track the size and number of pins.
John Q. **updates his directory** (adding and deleting some photos, changing names of others), and **re-uploads** it to Cube.	Cube interfaces with IPFS to add, delete, and rename files. The service replicates data such that there's **no downtime** while IPFS unpins and runs GC. [^4]	IPFS pins new files in the directory, removes pins and blocks that should no longer exist, runs GC, and creates a new hash for the root directory.

Notes

- [^2] In the medium term, we can look forward UnixFSv2 and selectors landing, which will make our lower-level architecture more effective, and obviate some of our existing user experience bottlenecks. This work, however, does not block the short-term actions laid out above.
- [^3] See this comment for more detail on Pinning Service API endpoints for payment types: https://github.com/ipfs/notes/issues/378#issuecomment-519125912
- [^4] "Part of [the challenge here] is just due to some of the limitations in how IPFS works in pinning. The scale we're running at -- tens of thousands of hashes per node -- can make content discovery difficult. This is mostly due to how we simply can't announce all of the content we have fast enough before content announcements expire. Some of this is rooted in challenges the DHT has with undialable nodes. We'll have to see how future IPFS updates effect this.

  The big problem is that GC on IPFS doesn't really work like a normal file system. When you delete something, it doesn't immediately go away. You have to "unpin it" and then run a garbage collection process to get rid of it. Right now, our nodes take roughly 10 hours to GC, and when that happens we can't pin. In the beginning, we got around it by ... not GC'ing. Now we replicate across multiple nodes and have to intelligently schedule garbage collections to make sure content is always online. This is a really tough problem to solve and as we scale this might not be the best solution."
- The relationship of IPLD selectors and third-party use of the Pin API: [link coming soon]
- IPFS Cube Product Proposal: https://docs.google.com/document/d/1yfef8xdpyeLXz_PQp3qofZ6tzEhDXfUZ0oEBBxfB3QQ/edit#
- Pinning Service API: https://github.com/ipfs/notes/issues/378
- IPFS Cluster <> Filecoin Integration Proposal: https://docs.google.com/document/d/1BUt7stI6gtIBuLrQYzNJ5hEdRaHrbU3a-Um7MfyYAYw/edit#
- Experiment in MFS-based cohosting: https://github.com/ipfs-shipyard/cohosting/pull/2

Features

1 IPFS-as-file-system (Desktop/WebUI)

North star

A non-developer end user should not need to know what 'pinning' is to get the benefits of IPFS. We should abstract the idea of 'pinning' away by re-branding MFS as "IPFS Drive" and build our GUI products as different interfaces into one user experience. **The CLI action `ipfs add` needs to align with changes in GUI.**

Priorities

In the short term, the low hanging fruit is to move our GUI applications and our CLI from using the low-level Pin API to adding user data to MFS[1], where all files are implicitly pinned, and are also much easier to manage via WebUI. We should also transition the language in our GUI apps (both visual and words) away from 'pinning' and towards clear, non-technical vocabulary.

> Version 0.1

After this release, a user can add objects to IPFS Desktop, WebUI, Companion, and command line and have the same experience (ie, see all files and folders listed on Files Page in GUI applications).

1. IPFS GUI applications and IPFS CLI add objects to MFS instead of using the low-level Pin API.
    a. Current behavior
        i. Companion: upload via right-click context menu or browser action menu item adds raw data to IPFS and it is up to the user to memorize or save CID to find it later in the Files Page/pins interface.
        ii. When users `ipfs add` a file via command line, it doesn't appear in our WebUI because is has not been `ipfs files cp`'ed to MFS as well. It just shows up as an unidentifiable QmHash in the Files Page/pins.
    b. Desired behavior
        i. Allow user to manage all files via WebUI's Files Page instead of split between Files Page/files and Files Page/pins.
        ii. Align CLI action `ipfs add` with changes in GUI so that WebUI reflects changes made via command line, and vice-versa.
    c. Implementation options
        i. When file is uploaded via the browser extension, it should automatically be added to MFS. It will then appear in Files Page/files automatically (could be

[1] With IPFS, all objects live under /ipfs, and everything under /ipfs is immutable. It's like /ipfs is a forest of everything, but not a place where you can manage those files. MFS is like a separate root that behaves like a file system. Every time you change things in this directory, all hashes -- everything that happens in the background -- is hidden. People don't need to know about that. Everything in MFS is implicitly pinned (ie, it won't be garbage-collected unless explicitly removed from MFS). So, it's a way to abstract away the low-level APIs in a way that's local, and intuitive.

added as /ipfs-companion-uploads/upload_YYYY-MM-YY_HHMMSS, for example, to provide a title, similar to how screenshots get saved).

ii. To match GUI's capability and ensure the same interaction with files across IPFS, we should disable pinning by default for `ipfs add`, and instead, add to MFS by default.

2. Overlap in function and meaning between IPFS Desktop+WebUI Files Page/files, Files Page/pins, and Explore/Inspect Page is resolved.
    a. Current behavior
        i. The Files Page has sub-pages for "files" and "pins," rather than only Files.
            1. "Files" == things added to MFS. They are displayed as objects a user expects to see (images, documents, folders, etc).
            2. "Pins" shows a list of hashes with no way to know what is in each hash. There is some overlap between what's shown in Files as a file, and what's shown under "pins," but the same QmHash in Files and Pins behaves differently (ie, can't perform the same actions on them in the different sub-pages; can't preview items from Pins sub-page).
        ii. In WebUI, there is an "Explore" Page. In Desktop, the icons in the left nav are unnamed. In both, when you are on the Files Page and select an action on an item, going to this view is called "Inspect." In all views, Explore/Inspect shows pins and blocks, but they are not clearly named as such.
    b. Desired behavior
        i. The Files Page only displays files and folders. A "file" or "folder" in our GUI apps always means the same as plain English: these are files and folders that may be made up of pinned blocks in the background.
        ii. The Files Page/pins sub-page is removed.
        iii. When users want to learn about or inspect IPFS implementation details, ie, the blocks and pins that make up "files" and "folders," they can find this information in the Inspect Page.
    c. Implementation options:
        i. Files Page
            1. Remove "pins" subpage
            2. Remove "pin" icon on Files Page; everything shown here, as it's in MFS, should automatically be pinned so it is not an option.
            3. Remove option to Pin/Unpin; "Delete" serves this user need, while unpinning in the background.
            4. Focus information in upper right area on files, rather than backend IFPS concerns, by:
                a. Removing information about "pins" and "blocks" from the upper right of files page.
                b. Keeping "files" and changing "repo" to "IPFS cache"[2]

[2] "files" == MFS
MFS is a subset of all blocks in "IPFS cache"
Non-MFS things in ""IPFS cache" are blocks cached when accessing other IPFS resources (browsing websites, ad-hoc fetch of remote data, etc.) aka "blocks transported thru your local node, but not added to MFS." This also includes low-level pins that were not added to MFS.

        c. Adding "about these numbers" expandable accordion as per this design to provide more explainer info on what's going on because it gets complicated. See also footnote 2.
    5. If a pin exists in MFS without a file name, display this pin as a file without a title, instead of two QmHashes. The 'title' can say "unnamed file" or similar.
        a. When copying a file to MFS, the API requires a name. Right now, we default to using QmHash. It is straightforward to add another step that does something like "if title == QmHash, replace title with "unnamed."
    6. Add 'copy hash' as an option via checkmark so that menu has the same options as the drop-down menu.

ii. Explore/Inspect Page

    1. Rename the Explore Page to the Inspect Page across all GUI applications. 'Inspect' is a clear word for 'the place you go to look into the internal details of a file.'
        a. Or, add a different entry point to Explore in addition to Inspect. There could still be a use case for an exploratory area for poking around at charismatic datasets.
    2. When a user selects "Inspect" from the "more..." action menu on, for example, a photo from their Files Page:
        a. They should see whatever title is associated with the item at the top of the page, rather than the QmHash. This could mean that at times "unnamed" is at the top of the page.
        b. They should understand where they are in the tree display and be able to get more detail about the pins and blocks that make up their file on IPFS.
            i. Right now, all nodes look the same. You can traverse a low-level view of directories, but you have no indication of where the file starts. It's just a continuous tree. We should:
                1. Use a different color/shape for those blocks in the tree that are pinned.
                2. Provide visual indicators and words showing where the root pin is, and how the root relates to the other low-level files/blocks/pins being displayed in the interface.

vvvvvvvvvvvvvvv everything below a separate release vvvvvvvvvvvvvvv

3. Get pinning abstractions and language right ("adding" / "putting" / "saving" to IPFS does what users expect without needing to understand the low-level concept of IPFS "pinning").

---

   a. Current behavior: Our "saving on IPFS" journeys often start with "I saw this thing on IPFS and want to save it." These journeys also don't support users' needs around data persistence and availability.
   b. Desired behavior: Instead, we should start with "I have a file on my local machine that I want to put on IPFS;" ie, saving is an 'upwards' motion 'to' to network, rather than a 'downwards' motion 'from' the network. Make clear the relationship of adding objects to IPFS and public/private; privacy expectations, removing/deleting objects, and the availability of objects.
   c. Implementation options:
      i. [TODO: what visuals/words to use for adding/saving/etc]
      ii. [TODO 'Adding' objects to IPFS <> public/private; privacy expectations]
      iii. [TODO 'Adding' objects to IPFS <> removing/deleting objects]
      iv. [TODO 'Adding' objects to IPFS <> availability of objects]

> Version 0.2

After this release, a user can intentionally choose to add objects via our IPFS GUI platform to the public IPFS network or their local, private IPFS peer.

1. Create Public/ and Private/ directories on IPFS Desktop and WebUI
   a. Current behavior: IPFS Desktop/WebUI/Companion are proxies to a local IPFS node, and by default, are part of the public IPFS network.
   b. Desired behavior: Anything that is added to Private/ MUST NOT be provided on the public DHT. Anything added to Public/ SHOULD be provided back to the public IPFS network.
   c. Implementation options:
      i. [TODO]
2. Create good user experience for Public/ and Private/ directories
   a. Current behavior: IPFS Desktop/WebUI/Companion do not explicitly guide users to understand the privacy or accessibility of data on their local IPFS nodes.
   b. Desired behavior: Users should understand the ramifications of adding content to Public/ or Private/ and be supported when they make mistakes (as humans do).
   c. Implementation options:
      i. [TODO]The user is supported through tips and interactions to understand what each directory means, and what to expect in terms of third-party accessibility to the objects inside. Via usability testing, designs must prove that users understand:
         1. Who can access/see objects in their Public/ directory
         2. Who can access/see objects in their Private/ directory
         3. What the difference is between the two directories

Goal: Share objects via links

Implementation:

1. Improve (via usability testing) the UI of the existing ability to share from Public/. Via usability testing, designs must prove that users can:
   a.
2. Develop UI for the experience of sharing from Private/

   a. Ask user if they want to copy file to Public/ before sharing
   b. Provide "encrypt and copy shareable IPFS link" functionality, which asks for a password before encrypting and copying data to Public/

> Version 0.3

Goal: Users can delete objects from their Private/ folders; ie, delete them entirely from their machines and from IPFS. Users can remove objects from their Public folders; ie, stop 'co-hosting' them, but not necessarily remove them from IPFS.

Implementation:

1. `Trash/` might work like `Private/` – blocks still in repo, still take space, but not announced. Removal from Trash removes blocks from repo, saves space. [Exact implementation tbd.]
2. Objects coming from Private/ should not be returned to people asking for them on the network. Objects coming from Public/ can be returned until they're GC'd.
3. UI has successfully-usability-tested 'delete' or 'remove' options. (The hope: the two slightly different 'delete'/'remove' processes will further support people's understanding of what you can expect in terms of privacy and data control on the dweb)

> Version 1.0 (MVP)

Goal: Launch an IPFS-based file system platform called "IPFS Drive" that provides a consistent user experience across web and OS.

Implementation:

1. [Backend: ?]
2. IPFS Drive has interaction, content, and visual design guides that support a consistent user experience across the platform.
3. IPFS Desktop, WebUI, and Companion implement these guides so that they look and feel like are interfaces into one user experience known as IPFS Drive.
4. IPFS Desktop, WebUI, and Companion are re-branded as different interfaces to one platform known as IPFS Drive.

> Version 1.1

Goal: Co-host → *see data persistence feature list below*

2 Data availability (Collaborative Clusters)

> Version 0.1

Goal: Users can easily join example Collaborative Clusters, understand why/how about Collaborative Clusters; and from admin perspective, create a 'followers' link.

Implementation:

1. Develop ability for a user to *create* a followers link in IPFS Cluster via the CLI (as there isn't a visual interface yet)
2. Develop an easy way for a third party running IPFS to *join* a Collaborative Cluster
   a. Click a button on a webpage; click a link anywhere
   b. Copy/interactions with button are usability tested for clarity

3. Spin up some Collaborative Clusters on IPFS storage with charismatic datasets as examples that people can 'follow'
4. Write marketing copy and visuals announcing 'new' IPFS capability

> Version 1.0 (MVP)

Goal: Users can see a basic admin panel when they spin up their Cluster.

Implementation:

1. Develop basic version of a visual admin panel.
    a. Can reuse the IPFS GUI visual language
    b. At this point, no interaction, just display of the information that's currently accessible via the Cluster CLI.
2. Develop hooks to power the visual display.

Goal: Users see prompts when using Cluster that help set intelligent system defaults.

Implementation:

1. Usability test Cluster setup workflow and re-write as needed, and add prompts as needed, that help users configure their Cluster in a way that is likely to work best for their specific use case.

> Version 1.1

Goal: User can control Cluster via admin panel, esp. add/delete peers, set replication factors, and set peer status (follower or full)
_Create follower link via admin panel
_Via admin panel, learn *why* they might make certain system choices/settings

Implementation:

1. _Visual admin panel expands to include interactions and Cluster control
2. _Interface so admin panel can control Cluster.
3. ...

> Version 1.2

Goal: _See performance and error logs via admin panel

Implementation:

1. _Visual admin panel includes performance and error reporting
2. _Hooks for visual display? Might already exist.
3. ....

> Version 1.3

Goal: Co-host → *see data persistence feature list below*

3 Data persistence (Pinning Services)

Version	Users can...	Frontend	Backend

V 0.1	_Understand how to persist data on the dweb.	_There are explanations and links pinning services and roll-your-own co-hosting information in IPFS GUI and Cluster applications.	n/a
V 0.2			
V 0.3			
V 1.0			
V 1.1			

Analytics / success metrics

Metric	Use case	Does metric collection mechanism exist today?
User engagement rates via opt-in reporting		
Number of interactions with “IPFS Drive” (IPFS Desktop and Companion) or IPFS Cluster per week	1️⃣2️⃣	⛔
Number of ‘shares’	1️⃣	⛔
Amount of data across peer(s)	1️⃣2️⃣3️⃣	?
Number and types of peers	2️⃣3️⃣	?
Number of accounts/users	3️⃣	⛔
Basic analytics		
Number of downloads	1️⃣2️⃣3️⃣	1️⃣ ✔️ 2️⃣ ? 3️⃣ ⛔
Opt-in share of error logs	2️⃣3️⃣	⛔
User frustration/happiness self-reporting		

"Was this helpful"-type questionnaire where appropriate, with open-ended box or issue for optional feedback	1️⃣2️⃣3️⃣	⛔
Number of pain points resolved for pinning services (removing friction from their workflows via technical changes/shipping features on IPFS)	3️⃣	⛔
Ease of task completion via usability tests		
Adding information to first, IPFS in general, and later, to Private and Public folders, and understanding what this means	1️⃣	⛔
Deleting information	1️⃣	⛔
Sharing information	1️⃣	⛔
Viewing/downloading information that has been shared with a user via IPFS or http	1️⃣	⛔
Adding and removing peers	2️⃣3️⃣	⛔
Creating follower links; adding and removing followers	2️⃣3️⃣	⛔
Being added as a follower (from the follower peer perspective)	2️⃣3️⃣	⛔
Adding and removing data from multi-peer systems	2️⃣3️⃣	⛔
Pinning-service-reported analytics (maybe they're willing to share aggregate information)		
Number of users	3️⃣	?
Number of users by amount stored (number of hashes and volume of data)	3️⃣	?
Amount of data pinned	3️⃣	?

Long-term vision

IPFS as a usable file system

- John Q. hears about this IPFS Drive (MFS) that works like Google Drive, but keeps his data secure, private, and in his control.
- He downloads IPFS Drive from the app store on his phone, and is able to move files into IPFS without knowing what a "pin" is. To John Q., it looks like he's sharing a picture the way he's used to; browsing to it on his phone, selecting it, hitting the 'share' button, and then selecting the IPFS app.
- He can also open the IPFS app directly and browse his files there, or add to his IPFS Drive from the app (which allows his to browse other files on his phone and select those he wants on his drive.)
- To manage his information, John can interface with his IPFS Drive via the app on his phone; via a browser; or via an app that he downloads to his desktop computer. In any interface, he can rename, move, and delete files. His changes are reflected on all platforms, and he understands what he is doing because the visual and written design choices are consistent throughout, and the interfaces are clear. [^1]
- Back to sharing those local pix with family. John Q. convinced his daughter to download IPFS Drive, too, so they can both have access to shared picture folders. Pictures John Q. shares in IPFS Drive are available to his daughter to browse.
- John Q. can also can share files with people who do not use IPFS. IPFS Drive allows you to share a permalink to every file or directory, which can be accessed via http.

Data availability and collaborative data stewardship

- Data Architect Maria works for a global environmental data and information service. She's always looking for ways to solve tricky problems in the large data archiving and access space.
- One of the problems that keeps her up at night is the sheer size of some of the newest satellite-derived information that her centers will need to archive and provide access to. The data is too large to go over normal network connections in any reasonable amount of time. Getting that information to scientists all over the world is a real challenge. (Not to mention the difficulties involved in running repeatable experiments on datasets when you can't identify and share them with confidence.)
- Maria hears about IPFS and its ability to move information in a peer-to-peer, content-addressed way. She also learns about IPFS Cluster, which can help her, as a data administrator, manage information across many peers in bulk.
- She tests it by spinning up a Cluster with peers across several data centers, which she does via command line because that's what she's used to.
- She adds a few large files to the system. She's used to large file ingest taking a bit of time. IPFS seems to handle this in about the same time as other similar systems.
- After she's added these files, Cluster lets her know that she can view and manage her peers via an admin panel.

- Via the admin panel, Maria learns about her options with regard to user and peer permissions, replication factors, and standard settings for different types of Clusters.
- Once Maria is confident that she understands the system, she runs a public beta. Network data access endpoints don't change, but Cluster now balances availability in the background, all the while collecting metrics and logs that Maria can use to fine-tune her systems.
- Next, she adds some of her science power users to the Cluster. She wants these scientists to be able to access the information they need directly, and also help support data availability by hosting information in their regions as well. The Cluster interface allows her to create a "Join my Cluster" link that she can send to these scientists. This will add their peers to her Cluster without giving them full admin access.
- Over the course of several years, Maria makes agreements with other data centers to support each others' information via IPFS. As their p2p stewardship practice matures, they've been able to realize the benefits of block-level de-dupe across their datasets, which has reduced the amount of information that needs to be stored and transported. This has really gotten at what Maria wanted: a better way to maintain and move massive datasets.

Data persistence and community co-hosting

- Alex, on top of her other librarian duties, happens to have inherited all local admin tasks, so it's up to her to keep the website lights on, etc.
- She learns about IPFS Cube, an out-of-the-box solution for IPFS community co-hosting. Quite literally out-of-a-box: you can order Cubes online, and they're not even very expensive. The local Code for America chapter that meets in a library conference room once a month can help her set it up at the next hack night.
- Once the little box is plugged in, its simple screen asks for a key to her IPFS Drive account so Alex can manage files from her phone or desktop computer.
- As a simple start, Alex connects her Cube's Public folder to the big screen at the library entrance. Pictures of local family and friends are set to rotate automatically. She's got plans for recorded family histories that people can view via 'checking out' a viewing pod at the library.
- Next, Alex is excited about publicising the library's new capabilities. She sends an email blast, and talks to people in the library, about joining the "Community Cube." All they have to do is go to an URL that she's created, and they can get approved to join. They can start adding their own family stories, and not have to worry about a for-profit website hosting service going out of business, and taking all of their data with it.
- As Alex's Cube experiment gets rave reviews, the library is able to purchase several more Cubes to provide better availability and redundancy for their own datasets, setting up the new Cubes as "followers" to the original. They really like buying these devices because they work out-of-the-box -- no set-up needed, and it works as soon as it's turned on.
- Other regional libraries follow Alex's lead, and buy their own self-hosted pinning services. They come to an agreement to help each other host data, providing even stronger availability and redundancy in case, for example, the power goes out at one location. They arrange themselves as a "federated" Cluster.

References

- Home base for work is here: https://github.com/ipfs/user-research/tree/master/pinning-ux
- For updates of work, follow "[EPIC] Pinning user experience": https://github.com/ipfs/user-research/issues/16
- For all issues related to this work, see tag "[Area] Pinning UX": https://github.com/ipfs/user-research/issues?q=is%3Aissue+is%3Aopen+label%3A%22%5BArea%5D+Pinning+UX%22

==========

SCRATCH PAD bits and pieces that might be useful later

- User Abbi wants to help co-host the Wikipedia page on quokkas because they love that little happy animal, so they flip the toggle in H. MacDev's app that says "Help co-host this page on IPFS." Abbi sees a progress bar that shows how much information is being downloaded locally and how long it's taking.
    - Design mandates: "co-host" to an end user should not be the same as "recursively pin everything in this directory," which can result in an end user accidentally pinning all of Wikipedia.
- User Baptista wants to see everything they've agreed to co-host. They use their co-hosting list like other people use pinboard.in, so they like to check in occasionally and add tags to make information more finable, and stop co-hosting sites that don't interest them anymore.
    - Design mandates: End users need to be able to understand what they're co-hosting and add/remove information without needing to know IPFS pinning internals.
- User Carole was really inspired by the new black hole images, and so wanted to show his support for NASA by co-hosting their data. He navigated to the raw image repository for the black hole, flipped the convenient toggle in his app that says "Help co-host this page in IPFS"...and noticed that the progress bar said he'd be done in 36 hours?! And that the information would eat a year's worth of pinning service fees??!! 😱He quickly hit the "cancel" button and decided to co-host some local water data from USGS instead.
    - Design mandates: End users need to be able to cancel in-flight pinning actions.