IPFS and Package Managers: What Now, What Next?

Unpacking the 2019 Goal

Alex Potsides
IPFS

IPFS Camp

Package Managers Special Interest Group

ANDREW NESBITT

@andrew

ALEX POTSIDES

@achingbrain

JESSICA SCHILLING

@jessicaschilling

MOLLY MACKINLAY

@momack2

Why Package Managers?

- IPFS is a **great fit** for package management
- Improving for package managers will **improve for everyone**
- Success with package managers produces a **blueprint for adoption**
- Package manager maintainers/consumers are an **important demographic**
- IPFS can **directly impact** the package manager ecosystem

Package Manager Goals

1. **Make IPFS work better,** for package managers and for many other use cases as well
2. Get the package manager maintainer community **excited** about the benefits of using IPFS in their ecosystems
3. Demonstrate to package consumers that IPFS is ***holistically better*** than current centralized package managers
4. Increase **awareness and engagement** with IPFS (either to build new tools, or contribute directly) among package manager users/package publishers

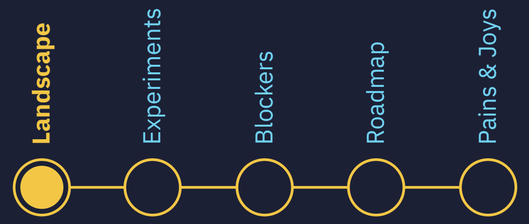
Package Manager Non-Goals

- Become the sole centralized maintainers of large new package manager infrastructure
- “Subvert” package manager maintainers or “steal control” from package creators
- ~~Implement~~ Ship YAPM (yet another package manager)

IPFS Camp

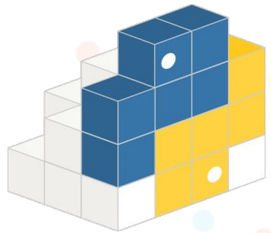
Package Managers:

Landscape



Package Managers





maven


CPAN


rpm



and many more ...

Package Manager Insights

- There are **very few** package manager maintainers in the world
- Package management is **code+infrastructure** for maintainers
- Maintainers **move slowly and carefully**
- It’s **hard to make money** in package management
- Infrastructure costs are **easily sponsored**
- **Network effects** in package management are very strong
- **Very little cross-pollination** between communities
- There are **new package managers** popping up every month

Clients

package.json

Registries

registry.npmjs.com

Verdaccio

JFrog
ARTIFACTORY

Package Formats & Manifests

npm: package.json

rubygems: rails.gemspec

pypi: setup.py

cargo: cargo.toml

maven: pom.xml

cpan: META.yml/META.json

go: go.mod

Redhat, Fedora etc: RPM

Alpine: APK

Gentoo: ebuild

Debian, Ubuntu: Deb

chocolatey: nupkg

Ubuntu: span

Arch: Pacman

System Package Managers




Language Package Managers









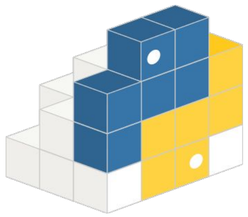


Categorizing Package Managers

- **Centralized registry**
  Client, tooling and community focused primarily on a single registry and namespace for all packages; public mirrors and alternative registries are infrequent
- **Portable registry**
  Similar to centralized, but users keep a local copy of the whole registry (often as a git repo) which is regularly synced by users
- **Multi-registry**
  Clients are designed to be used with multiple primary registries; users have choices of multiple registries to publish to; mirrors are frequently used
- **Registry-less**
  No concept of a registry or namespace where packages are published to; clients directly link to tarballs or git repositories on any URLs, resolved with DNS

Centralized Registry






Multi-Registry




Portable Registry



Registry-less



Categorizing for Integration

- **File system-based** (maps closely to multi-registry category)
  Literally a network attached folder full of files and other folders; metadata is also stored as files, so everything is easily mirrored
- **Database-based** (maps closely to centralized registry category)
  These run a database-backed web app for authentication, uploading and providing APIs for clients; mirroring is therefore tougher, and APIs are usually unique
- **Git-based** (using portable registries)
  These use Git or GitHub as a database rather than a traditional centralized database
- **Git-based** (registry-less)
  No hosted registries at all, instead declaring dependencies as fully qualified HTTP URLs or GitHub URL shortcuts; this means thousands of single-project registries that rely on Git/GitHub for release metadata

File System-Based




Database-Based





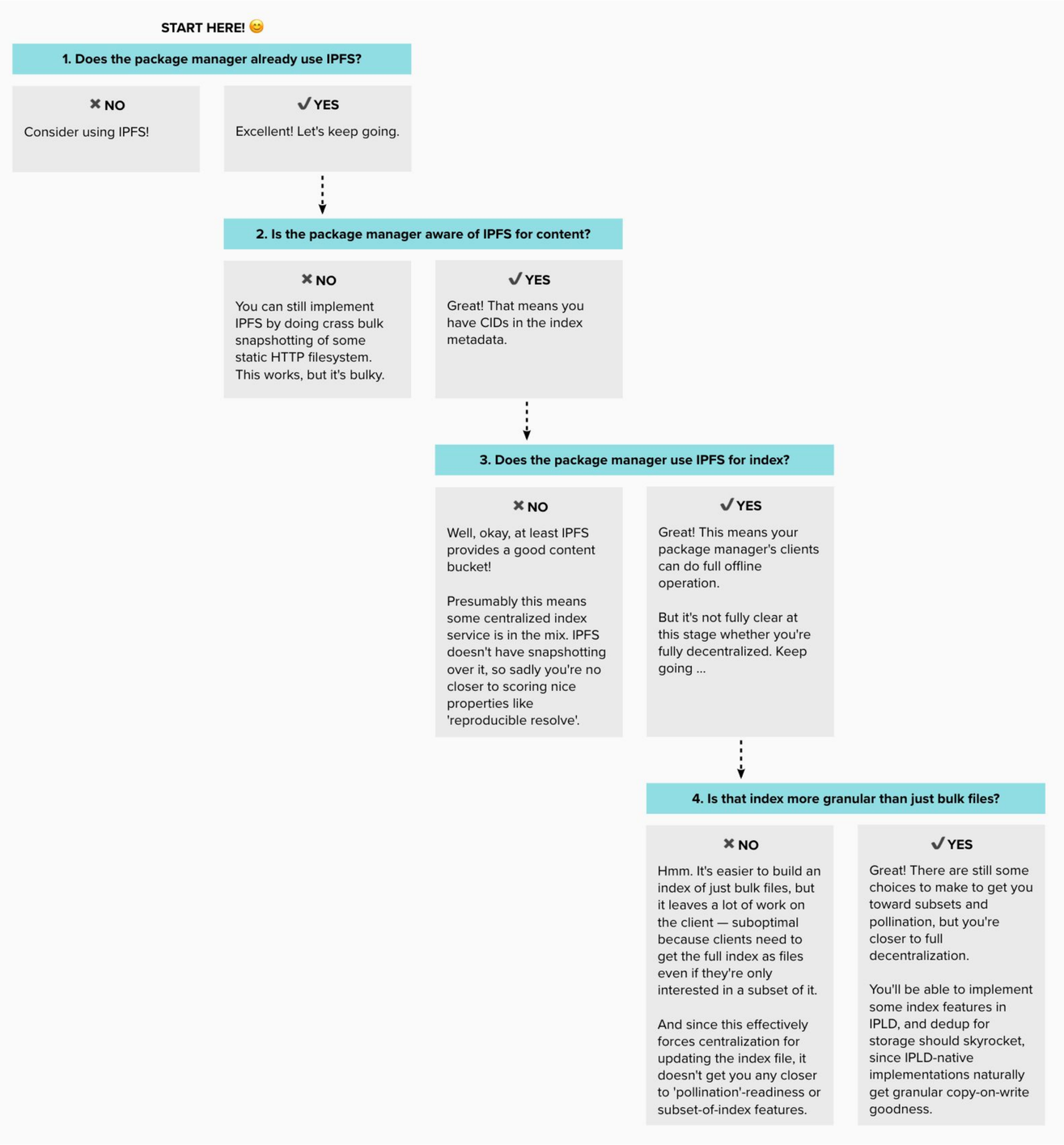

Git-Based



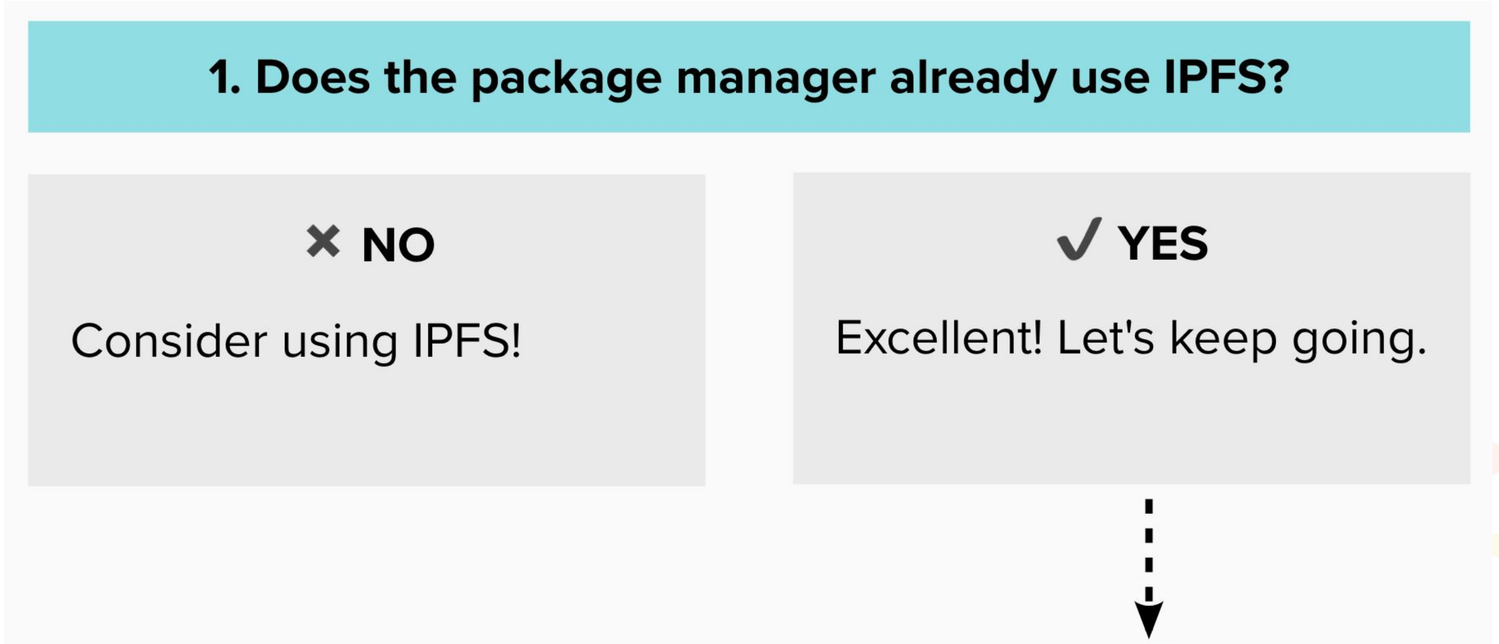




Depths of Integration: Decision Tree



Depths of Integration: Decision Tree



Depths of Integration: Decision Tree

2. Is the package manager aware of IPFS for content?

✖ NO

You can still implement IPFS by doing crass bulk snapshotting of some static HTTP filesystem. This works, but it's bulky.

✓ YES

Great! That means you have CIDs in the index metadata.

Depths of Integration: Decision Tree

3. Does the package manager use IPFS for index?

✖ NO

Well, okay, at least IPFS provides a good content bucket!

Presumably this means some centralized index service is in the mix. IPFS doesn't have snapshotting over it, so sadly you're no closer to scoring nice properties like 'reproducible resolve'.

✓ YES

Great! This means your package manager's clients can do full offline operation.

But it's not fully clear at this stage whether you're fully decentralized. Keep going ...

Depths of Integration: Decision Tree

4. Is that index more granular than just bulk files?

✖ NO

Hmm. It's easier to build an index of just bulk files, but it leaves a lot of work on the client — suboptimal because clients need to get the full index as files even if they're only interested in a subset of it.

And since this effectively forces centralization for updating the index file, it doesn't get you any closer to 'pollination'-readiness or subset-of-index features.

✓ YES

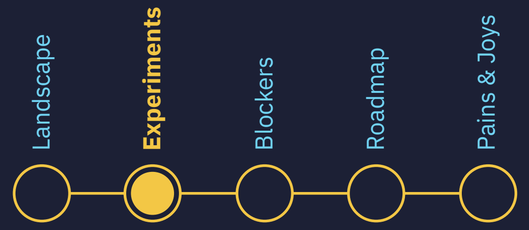
Great! There are still some choices to make to get you toward subsets and pollination, but you're closer to full decentralization.

You'll be able to implement some index features in IPLD, and dedup for storage should skyrocket, since IPLD-native implementations naturally get granular copy-on-write goodness.

IPFS Camp

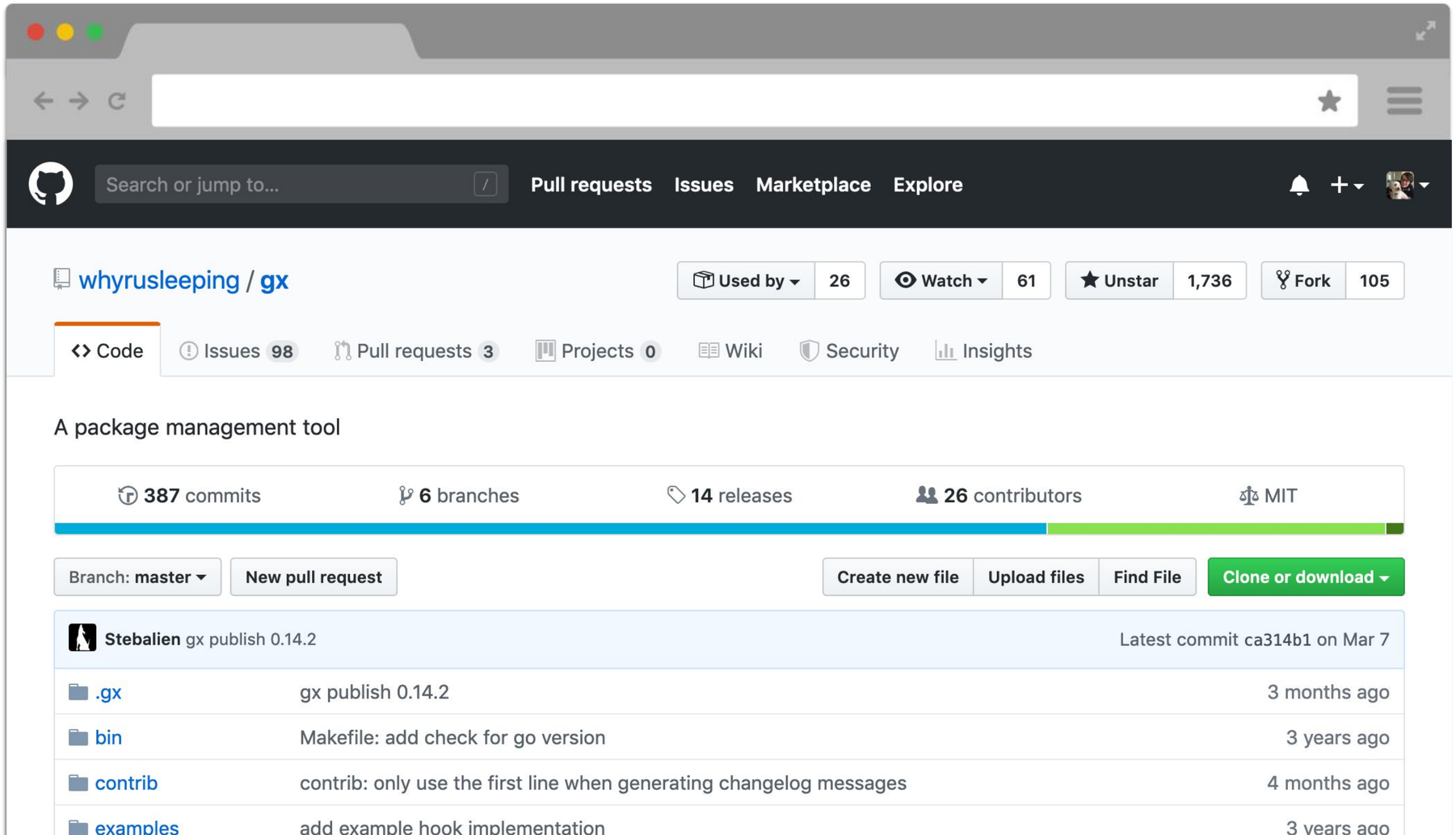
Package Managers:

Experiments



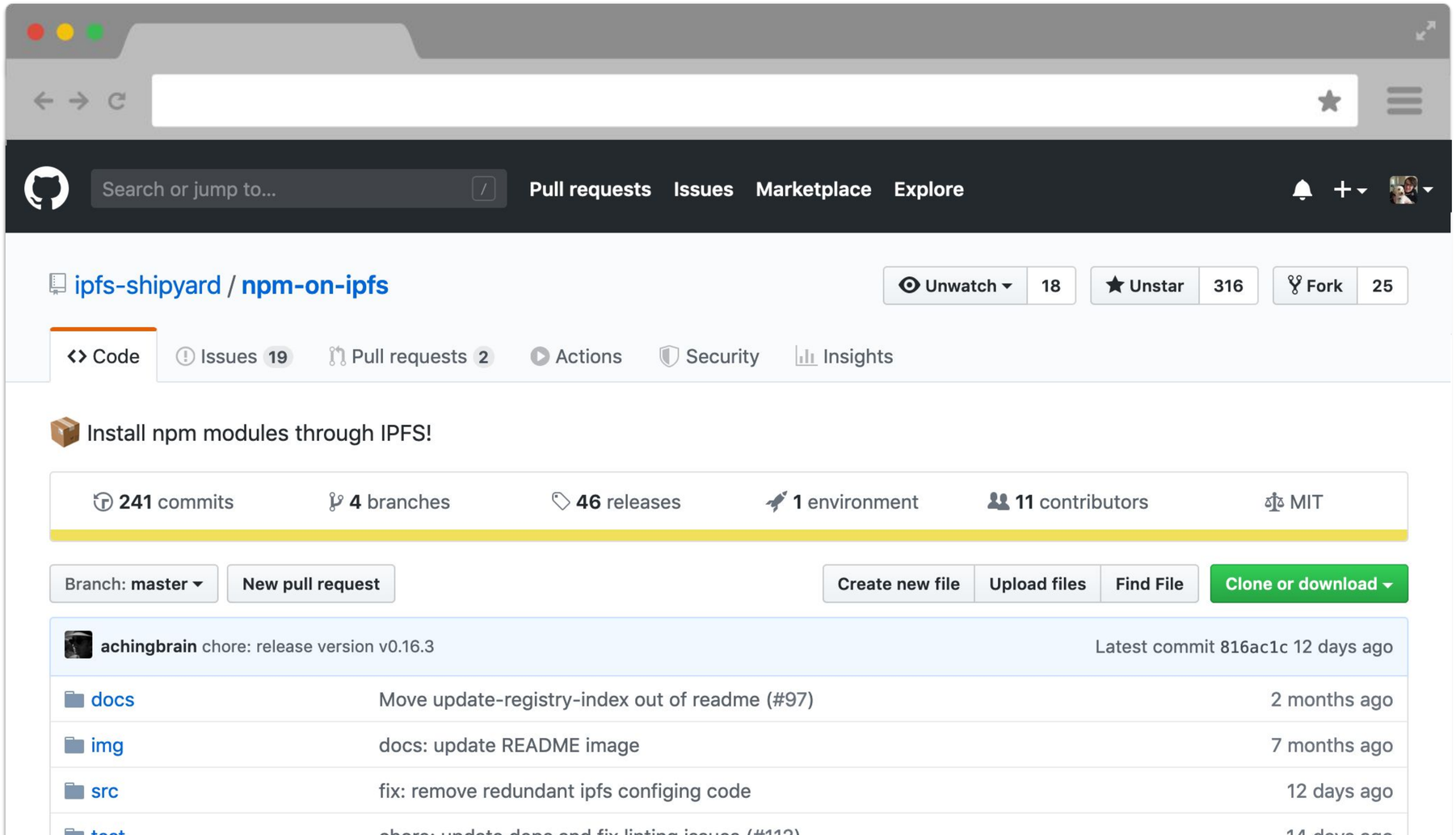
Experiment:
gx

- Content Addressable ✔️
- Decentralized ✔️
- Reproducible Builds ✔️
- Hashes in source code 🧐
- Transitive upgrade UX 😤
- Go Modules 🧨



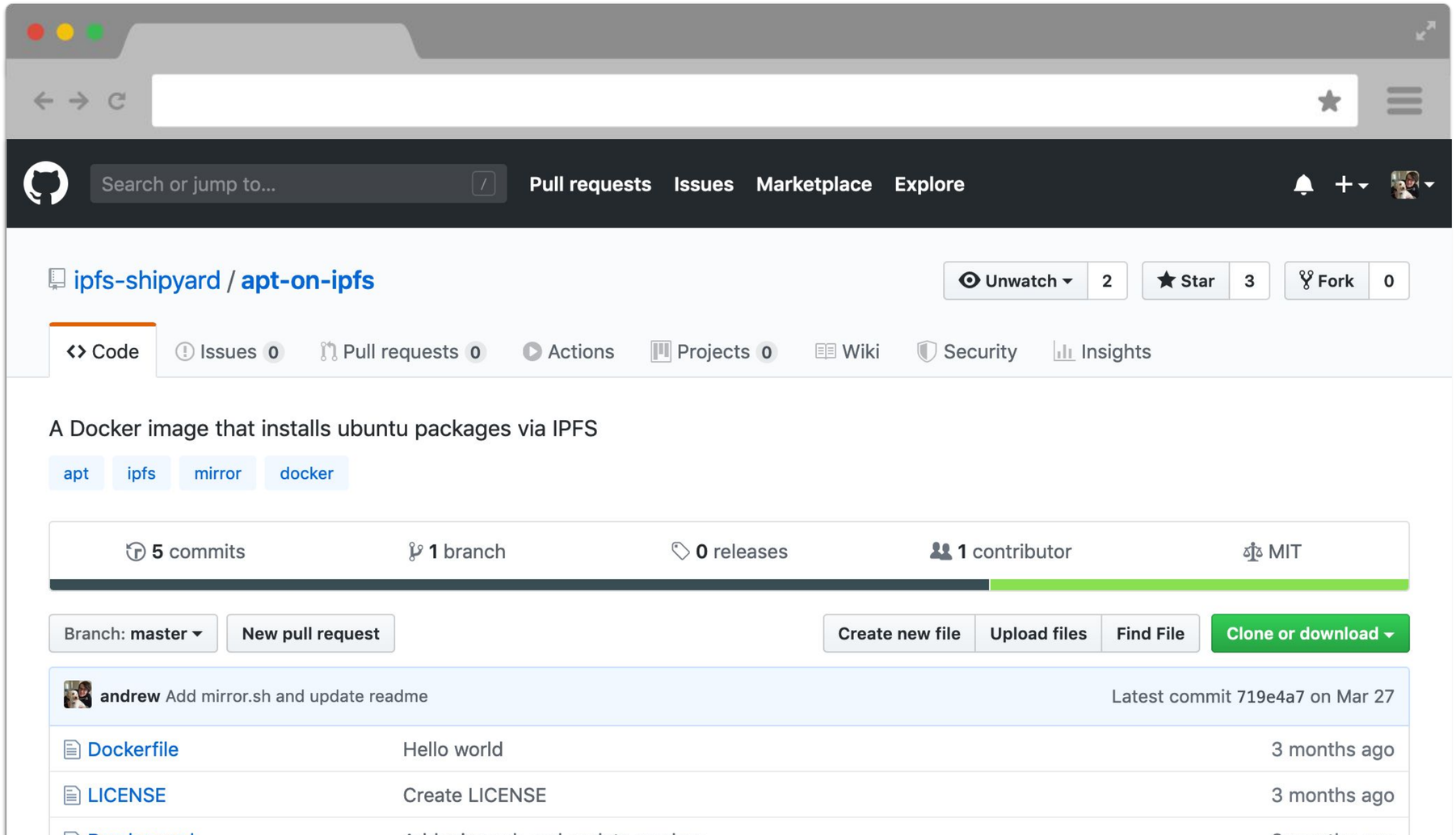
Experiment:
npm-on-ipfs

- Offline Friendly ✔️
- Big Data on IPFS ✔️
- MFS in production ⭐
- js-ipfs testbed 👩‍🔬
- Centralized Server 🙈
- Performance 😤



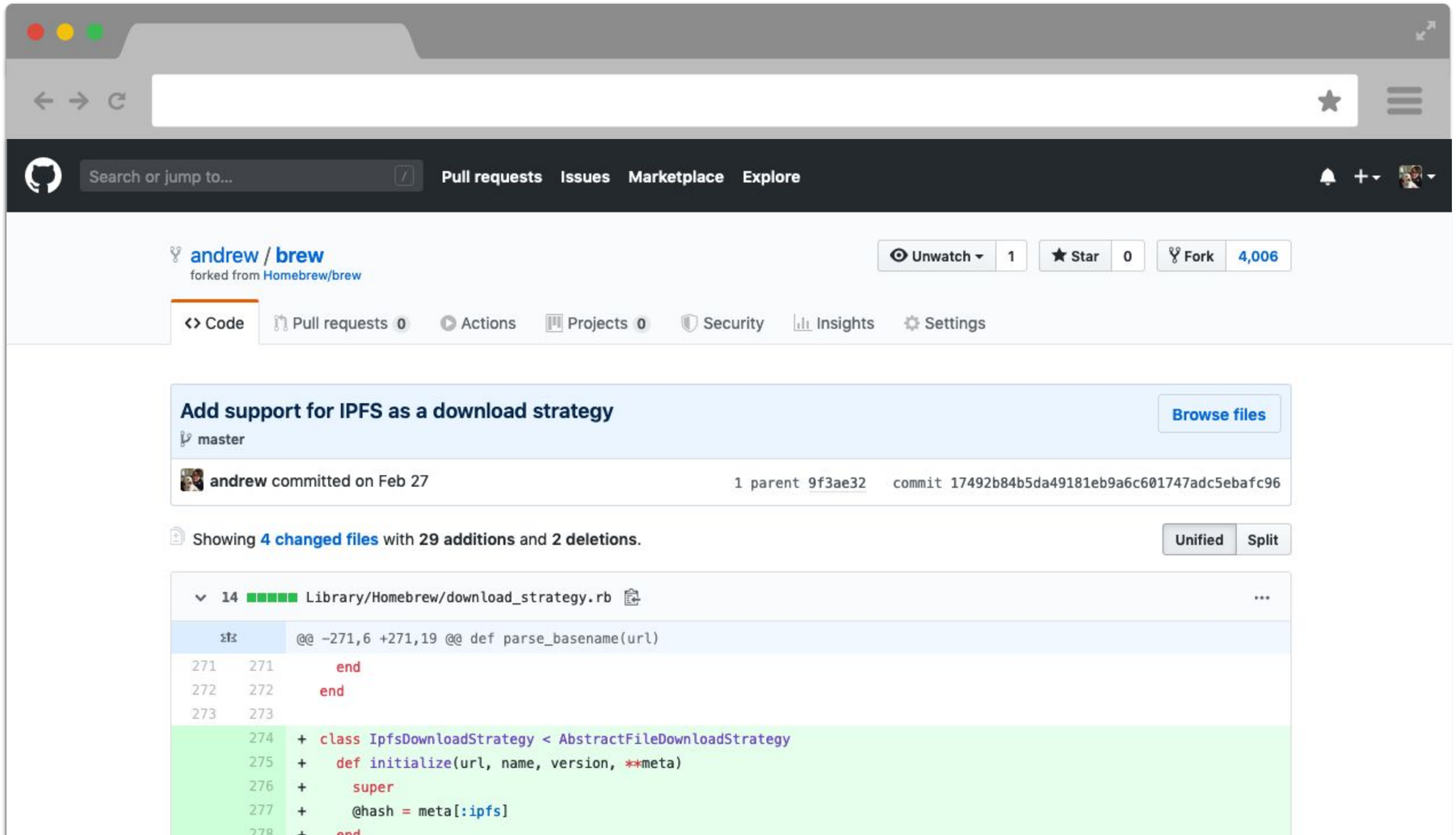
Experiment:
apt-on-ipfs

- Self-Service ✔️
- Mirrors the Mirrors ✔️
- Very slow setup 😴
- Can’t use new features 😢
- Resource intensive 🚛



Experiment:
homebrew-on-ipfs

- Just another transport ✔️
- Adds offline support ✔️
- Hashes in the index 🤩
- brew install ipfs 🚀
- Unlikely to be default 😢



IPFS Camp

Package Managers:

Blockers

Landscape
Experiments
Blockers
Roadmap
Pains & Joys

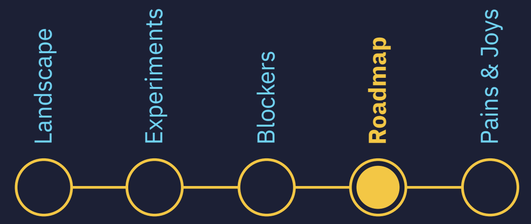
Current Blockers to Adoption

- **Requires 2x disk space for mirroring**
  Filestore expects files to be immutable once added, so rsyncing updates to existing files causes errors
- **No easy way to directly add a directory to MFS with go-ipfs**
  Adding a directory of files to MFS means calling out to `ipfs files write` for every file; ideally there should be one command to write a directory of files to MFS
- **Updating rolling changes requires rehashing all files**
  If there is a regular cron job downloading updates to a mirror with rsync, there's currently no easy way to only re-add the files that have been added/changed/removed without rehashing every file in the whole mirror directory

IPFS Camp

Package Managers:

Roadmap



Roadmap in Brief

- **Q3:** Improve introspection, debuggability, testing, benchmarks to simplify identifying issues affecting IPFS usability
- **Q3-Q1:** Improve scalability and performance
- **Q3:** Improve docs, guides, and dev experience (APIs) to make creating package manager POCs with IPFS clear, easy and fast
- **Q3:** Through POCs, demonstrate tangible present value for how IPFS can improve over centralized systems
- **Q3-Q1:** As progress is made, document improvements and growth trajectory on tests, benchmarks, and usability
- **Q4:** Empower communities to migrate IPFS and gain tangible benefits, thus creating momentum
- **Q4-Q1:** Invest embedded resources in collabs and partnerships with package manager maintainers to reduce barrier to entry

Focus:
Performance

1. **Make IPFS as a content store the clear choice for database package managers**
   - No major performance snags
     - IPNS
     - MFS
     - Fast adding/updating
   - Works well with existing tooling (SHA hashes)
   - POCs prove out the main components
   - Good metaphors, UIs and visualizations to make sense of how IPFS package managers work
   - Scaffolded onboarding experience (desktop as onramp?)

2. **Make IPFS the easiest tool for mirroring file system package managers**
   - Targeted documentation/guides
   - Simple, consistent, documented commands
   - Adding/updating performance
   - POCs prove out the main components
   - Point-to-point mirroring benchmarks

Focus:
Pipeline

1. **Make it easier to build IPFS-powered package manager POCs**
   - Easier API with optimal defaults and good documentation
   - High-level guides and explainers
   - Better debuggability and testing

2. **Improve tooling to identify/rank/fix issues identified in POCs**
   - Package manager benchmarks and key metrics
   - Package manager testing harness/introspection/debug tools
   - Public dashboard highlighting progress on issues

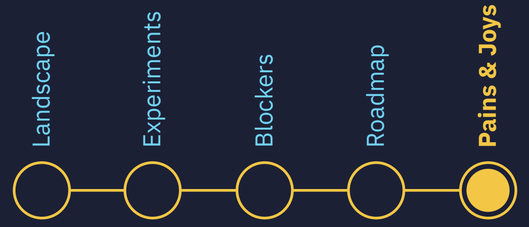
3. **Create POCs to stress-test/benchmark IPFS and highlight gains**
   - Speed to set up a mirror and propagate mirror updates
   - Speed to sync content across the network
   - Size of mirror
   - Size of dependency tree when deduped using IPFS

4. **Propagate improvements to package manager ecosystems**
   - Hands-on, embedded collab with key maintainers
   - Socialise POCs, highlighting relative value/cost

IPFS Camp

Package Managers

Activity: Pain Points & Benefits



Insights:
Package Manager Problems IPFS Can Solve

1. **Dependencies**
   - Not being able to reproduce a known working set of dependencies at a later date
2. **Security, verifiability and trust**
   - Can’t confirm package contents are as originally published
   - Hard to coordinate key-signing infra between maintainers
3. **Reliability and availability**
   - Package releases get removed from registries
   - Can’t install or build packages when you’re offline
4. **Cost**
   - Hosting internal/private mirrors costs in both time and maintenance/infrastructure

Insights:

Benefits of IPFS-Enabled Package Managers

1. **Ease of use**
   - Easier (and therefore less costly) to mirror a registry
2. **Resource savings**
   - Save bandwidth by reducing re-downloads
   - Minimise storage requirements by de-duping contents
3. **Performance**
   - Faster to download from IPFS-powered registries/mirrors
4. **Offline availability**
   - Better resilience to network problems during installations
   - Possible to install and verify packages when offline
5. **Reproducibility**
   - Easy to declare/verify the provenance of everything you run
6. **Forkability**
   - Users can easily fork existing registries
   - Users can easily start their own registries
7. **User-hosted**
   - Users can replicate and provide packages they care about
   - Users automatically cache items that they rely on
8. **Decentralised publishing**
   - Community doesn’t need to rely on centralised registries
   - Users can publish directly to IPFS
   - Not as many central points of failure — or of trust

Now it’s your turn

THANK YOU!