Nix × IPFS

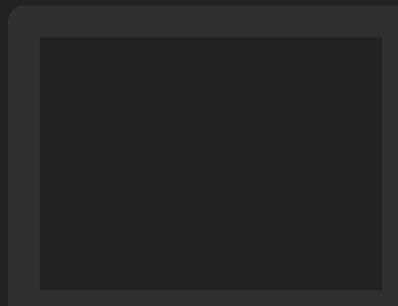
NixCon 2020

JOHN.ERICSON@OBSIDIAN.SYSTEMS

What is IPFS?

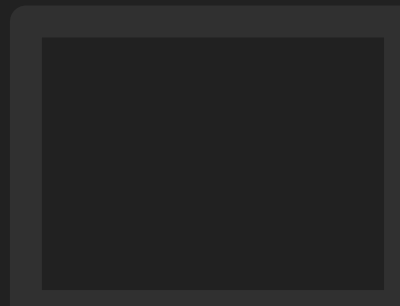
What is IPFS

```
$ echo '{"hi": ""}' | ipfs dag put
Bafyreidk...l3n53dv7w2z
```



What is IPFS

```
$ echo '{"hi": ""}' | ipfs dag put
Bafyreidk...l3n53dv7w2z

$ ipfs dag get Bafyreidk...l3n53dv7w2z
{"hi":""}
```



What is IPFS

```
$ echo '{"hi": {"/": "Bafyr...7w2z"}}' | ipfs dag put
Bafyreid3...spdhu6jixba

$ ipfs dag get Bafyreid3...spdhu6jixba
{"hi":{"/":"Bafyreidk...l3n53dv7w2z"}}

$ ipfs dag get Bafyreid3...spdhu6jixba/hi
{"hi":""}
```

What is IPFS

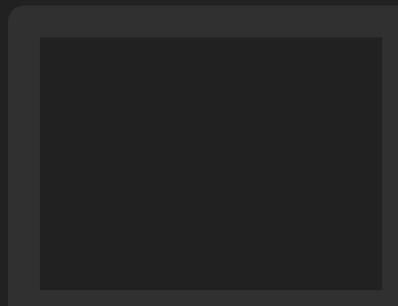
```
$ echo '{"hi": {"/": "Bafyr...w2z"}}' | ipfs dag put
Bafyreid3...spdh
```

Called IPLD link

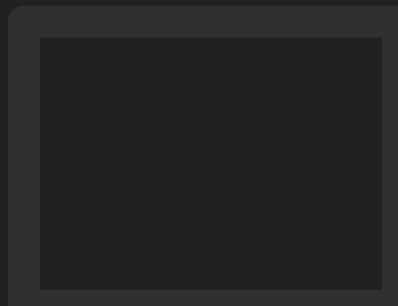
```
$ ipfs dag get Bafyreid3...spdhu6jixba
{"hi":{"/":"Bafyreidk...l3n53dv7w2z"}}

$ ipfs dag get Bafyreid3...spdhu6jixba/hi
{"hi":""}
```

What is IPFS

- Data model called IPLD
- Big tent approach — support popular existing formats
- …including git

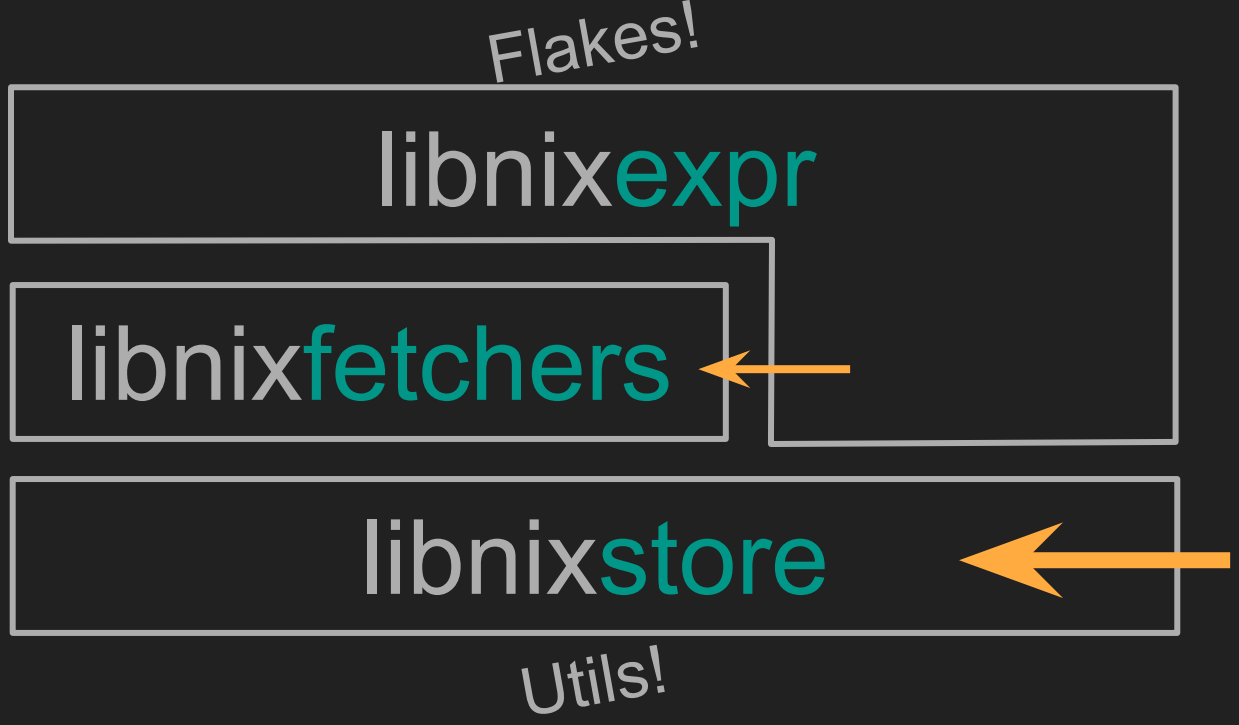


What’s Inside Nix?



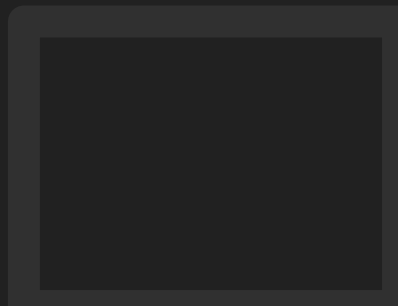
Layers of Nix

Layers of Nix



libnixstore types: `StorePath`

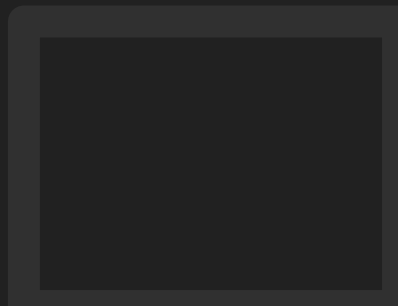
/nix/store/nmgiyxa9fpn42w6xjd1wvzgb1svffn1w-bash



libnixstore types: `StorePath`

/nix/store/nmgiyxa9fpn42w6xjd1wvzgb1svffn1w-bash

name

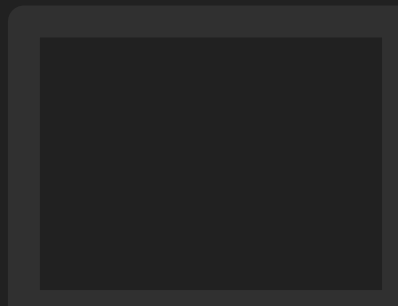


libnixstore types: `StorePath`

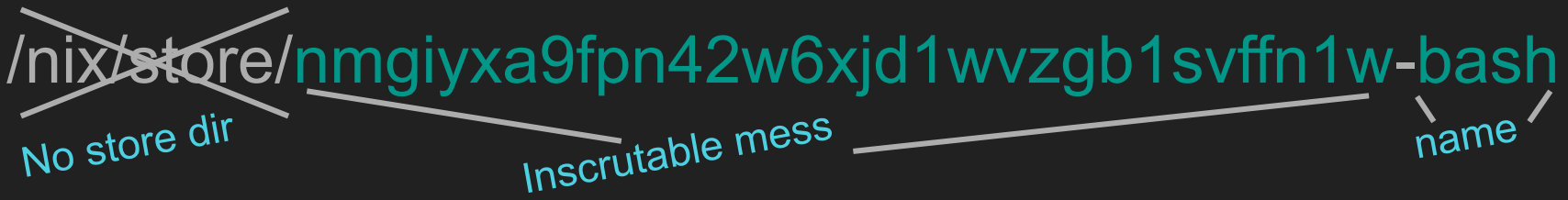
/nix/store/nmgiyxa9fpn42w6xjd1wvzgb1svffn1w-bash

Inscrutable mess

name



libnixstore types: `StorePath`



`StorePath` Provenance

- Two choices:
  - Input addressed: How made
  - Content Addressed: What is
- Can verify but not derive

libnixstore types: `Derivation`

- Outputs
- Builder
- Input sources
- Input derivations
- …

libnixstore types: `DerivationOutput`

```
struct DerivationOutput
{
    StorePath path;
    std::string hashAlgo;
    std::string hash;
}
```

libnixstore types: `DerivationOutput`

```
struct DerivationOutput
{
    StorePath path;
    std::string hashAlgo;
    std::string hash;
}
```

libnixstore types: `DerivationOutput`

```
struct DerivationOutput
{
    StorePath path;
    std::string hashAlgo;
    std::string hash;
}
```

libnixstore types: `DerivationOutput`

```
struct DerivationOutput
{
    StorePath path;
    std::optional<FileSystemHash> hash;
}
struct FileSystemHash {
    FileIngestionMethod method;
    Hash hash;
}
```

libnixstore types: `DerivationOutput`

```
struct DerivationOutput
{
    StorePath path;
    std::optional<FileSystemHash> hash;
}
struct FileSystemHash {
    FileIngestionMethod method;
    Hash hash;
}
```

libnixstore types: `FileIngestionMethod`

```
bool recursive;
```

libnixstore types: `FileIngestionMethod`

```
enum struct FileIngestionMethod {
    Flat,
    Recursive, // NAR
} method;
```

libnixstore types: `ValidPathInfo`

```
struct ValidPathInfo {
    StorePath path;
    Hash narHash;
    StorePathSet references;
    std::string ca;
    …
};
```

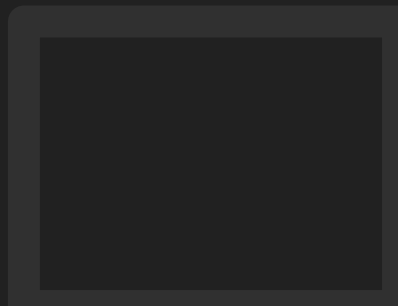
libnixstore types: `ValidPathInfo`

```
struct ValidPathInfo {
    StorePath path;
    Hash narHash;
    StorePathSet references;
    std::optional<ContentAddress> ca;
    …
};
```

New Feature #1:

Floating Content Addressed Derivations

(RFC #62 — thanks @regnat!)



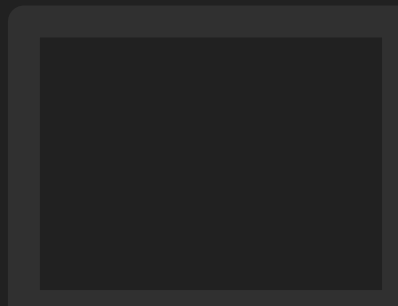
2 types of derivations: Then

Input Addressed:

- Outputs input addressed
- Pure
- Contents “floating”

Fixed Output:

- Outputs content addressed
- Impure
- Contents fixed



3 types of derivations: Now

Input Addressed:

- Outputs input addressed
- Pure
- Contents “floating”

Floating C.A.:

- Outputs content addressed
- Pure
- Contents “floating”

Fixed C.A.:

- Outputs content addressed
- Impure
- Contents fixed

3 types of derivations: Now

Input Addressed:

- Outputs input addressed
- Pure
- Contents “floating”

Floating C.A.:

- Outputs content addressed
- Pure
- Contents “floating”

Fixed C.A.:

- Outputs content addressed
- Impure
- Contents fixed

Best of both worlds

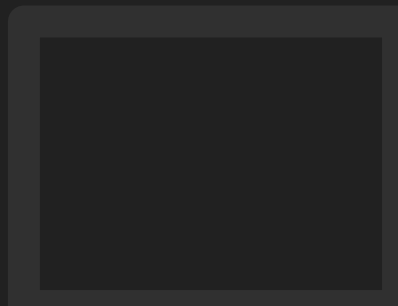
libnixstore types: `DerivationOutput`

```
struct DerivationOutput
{
    std::variant<
        DerivationOutputInputAddressed,
        DerivationOutputCAFixed,
        DerivationOutputCAFloating
    > output;
};
```

Why do for IPFS?

Why do for IPFS?

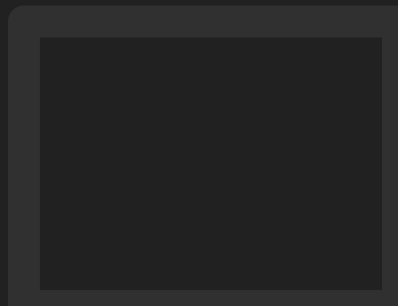
Answer: Content-addressing throughout



New Feature #2:

Git hashing

Tree & Blob, not Commit



libnixstore types: `FileIngestionMethod`

```
enum struct FileIngestionMethod {
    Flat,
    Recursive, // NAR
    Git,
} method;
```

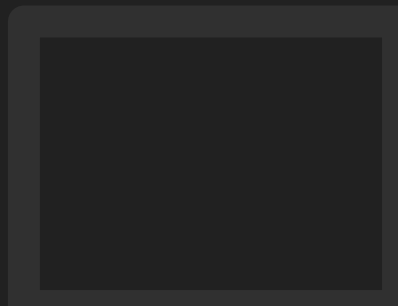
Git hashing benefits

- Fixed output from regular info
  - Before just: `nix-prefetch-git`
  - Now also: `git rev-parse <ref>:` (note colon!)
- `libnixfetchers` Also given support
- IPFS content ID ↔ Nix git CA path
  - Trustless translation!

New Feature #3:

C.A.-based queries

Move over, `StorePath`



libnixstore types: `StorePathDescriptor`

```
struct StorePathDescriptor {
    std::string name;
    ContentAddressWithReferences info;
};
```

libnixstore types: `StorePathDescriptor`

```
struct StorePathDescriptor {
    std::string name;
    ContentAddressWithReferences info;
};
```

= CA path provenance, decoded!

libnixstore types: `ValidPathInfo`

```
struct ValidPathInfo {
    StorePath path;
    Hash narHash;
    StorePathSet references;
    These<Hash, ContentAddress> narHashCA;
    …
};
```

libnixstore types: `ValidPathInfo`

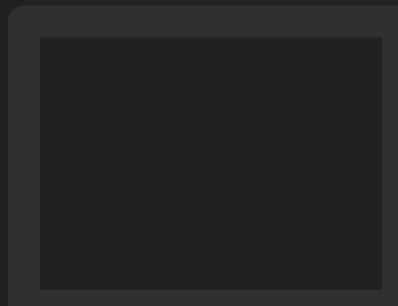
```
struct ValidPathInfo {
    StorePath path;
    ~~Hash narHash;~~
    StorePathSet references;
    These<Hash, ContentAddressHashCA;
    ...
};
```

These = one, the other, or both

O.S

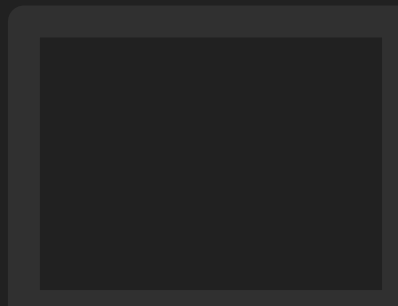
Store method refractors: before

- Query with store path
  -
- Path info always has nar hash
  -



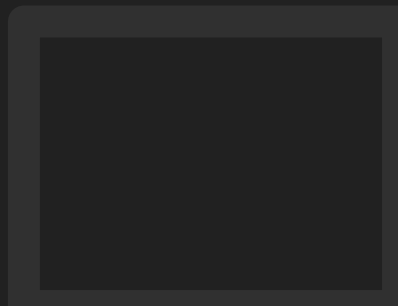
Store method refractors: after

- Query with store path
  - → Query with store path or descriptor
- Path info always has NAR hash
  - → Path info has at least one hash



Store method refractors: after

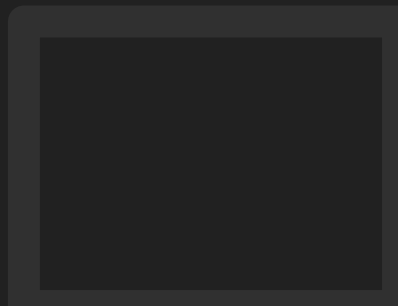
- Query with store path
  - → Query with store path or descriptor
- Path info always has NAR hash
  - → Path info has at least one hash
- Upshot: IPFS as trustless substitutor



New Feature #4:

“Git with references” convention

Not just for sources, anymore



Git with references example

```
ipfs dag get f0171122025ddb2f2ff6a6bbfe57a1137ad15d3e0ad068cb137e1570501c7ddcc24c19ebc | jq
{
  "cid": {
    "/": "baf4bcfgi2up65zpzhg2fmyi52kecqfhsaevbaha"
  },
  "name": "hello-2.10",
  "qtype": "ipfs",
  "references": {
    "references": [
      {
        "cid": {
          "/": "bafyreigpjalsm7jiyevkroghpnsmkh3lx3apymlxrdunc4fssmdv2patwy"
        },
        "name": "glibc-2.30"
      }
    ],
    "zhasSelfReference": true
  }
}
```

Actual file data

Git with references example

```
ipfs dag get f0171122025ddb2f2ff6a6bbfe57a1137ad15d3e0ad068cb137e1570501c7ddcc24c19ebc | jq
{
  "cid": {
    "/": "baf4bcfgi2up65zpzhg2fmyi52kecqfhsaevbaha"
  },
  "name": "hello-2.10"
  "qtype"
  "refer
    "refe ences": [
      {
        "cid": {
          "/": "bafyreigpjalsm7jiyevkroghpnsmkh3lx3apymlxrdunc4fssmdv2patwy"
        },
        "name": "glibc-2.30"
      }
    ],
    "zhasSelfReference": true
  }
}
```

Name for self-ref

Actual file data

Git with references example

```
ipfs dag get f0171122025ddb2f2ff6a6bbfe57a1137ad15d3e0ad068cb137e1570501c7ddcc24c19ebc | jq
{
  "cid": {
    "/": "baf4bcfgi2up65zpzhg2fmyi52kecqfhsaevbaha"
  },
  "name": "hello-2.10"
  "qtype"
  "refer
    "refe ences": [
      {
        "cid": {
          "/": "bafyreigpjalsm7jiyevkroghpnsmkh3lx3apymlxrdunc4fssmdv2patwy"
        },
        "name": "glibc-2.30"
      }
    ],
    "zhasSelfReference": true
  }
}
```

Name for self-ref

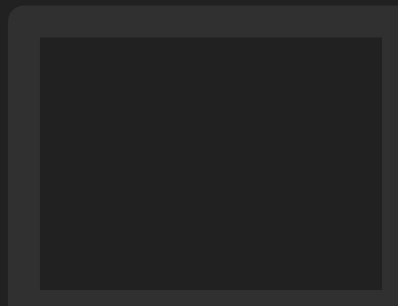
Actual file data

Likewise encoded references

New Feature #5:

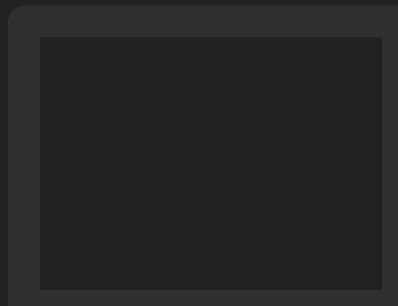
IPLD Derivations

Don’t forget the plan



IPLD Derivations

- Floating Content Addressed?
- Input sources only git (with/without refs)?
- Input drvs only other such drvs?
- If so: can convert to/from IPLD
  - All references as IPLD links!



IPLD Derivations example

ipfs dag get f017112201fb9b75473bb2c80744a1c472e6df89aef924c0d024d1a4b3ed0fb18fc26088a


```
{
  "args": [ "-c", "eval \"$buildCommand\"" ],
  "builder": "/nix/store/v0qd217nvhgws2w3id7xp6444lg3yf6d-dash-dir/dash",
  "env": {
    "buildCommand": "…bashbashbashbashbash…",
    …,
    "outputHashAlgo": "sha256", "outputHashMode": "ipfs", "system": "x86_64-linux"
  },
  "inputDrvs": [
    [ { "cid": { "/": "bafyreifdx57lk6pxmpp27hkgiakazx5cab2xcgxe2mse42ql3bx4thk6lm" }, "name": "root" }, [ "out" ] ]
  ],
  "inputSrcs": [
    { "cid": { "/": "bafyreifbamvfcut65kzcbanllirua3a5n2sqj4rcdduypbh5sclrmdk3cq" }, "name": "dash-dir" },
    { "cid": { "/": "bafyreigqh3k7cu2pbbjiue72bmw65ph6opvhkwrvuvwkdqupvopaqaqu6u" }, "name": "miniMkdir-dir" }
  ],
  "name": "dependent",
  "outputs": [ "out" ],
  "platform": "x86_64-linux"
}
```


O.S

IPLD Derivations example

ipfs dag get f017112201fb9b75473bb2c80744a1c472e6df89aef924c0d024d1a4b3ed0fb18fc26088a


```
{
  "args": [ "-c", "eval \"$buildCommand\"" ],
  "builder": "/nix/store/v0qd217nvhgws2w3id7xp6444lg3yf6d-dash-dir/dash",
  "env": {
    "buildCommand": "…bashbashbashbashbash…",
    …,
    "outputHashAlgo": "sha256", "outputHashMode": "ipfs", "system": "x86_64-linux"
  },
  "inputDrvs": [
    [ { "cid": { "/": "bafyreifdx57lk6p…                    …f2qi3bx4thk6lm" }, "name": "root" }, [ "out" ] ]
  ],
  "inputSrcs": [
    { "cid": { "/": "bafyreifbamvfcut65kzcbanllirua3a5n2sqj4rcdduypbh5sclrmdk3cq" }, "name": "dash-dir" },
    { "cid": { "/": "bafyreigqh3k7cu2pbbjiue72bmw65ph6opvhkwrvuvwkdqupvopaqaqu6u" }, "name": "miniMkdir-dir" }
  ],
  "name": "dependent",
  "outputs": [ "out" ],
  "platform": "x86_64-linux"
}
```


Links to input drvs

IPLD Derivations example

ipfs dag get f017112201fb9b75473bb2c80744a1c472e6df89aef924c0d024d1a4b3ed0fb18fc26088a


```
{
  "args": [ "-c", "eval \"$buildCommand\"" ],
  "builder": "/nix/store/v0qd217nvhgws2w3id7xp6444lg3yf6d-dash-dir/dash",
  "env": {
    "buildCommand": "…bashbashbashbashbash…",
    …,
    "outputHashAlgo": "sha256", "outputHashMode": "ipfs", "system": "x86_64-linux"
  },
  "inputDrvs": [
    [ { "cid": { "/": "bafyreifdx57lk6p…                    …z4qi3bx4thk6lm" }, "name": "root" }, [ "out" ] ]
  ],
  "inputSrcs": [
    { "cid": { "/": "bafyreifbamvfcut65kzcbanllirua3a5n2…                    …me": "dash-dir" },
    { "cid": { "/": "bafyreigqh3k7cu2nbb…                    …qqpvopaqaqu6u" }, "name": "miniMkdir-dir" }
  ],
  "name": "dependent",
  "outputs": [ "out" ],
  "platform": "x86_64-linux"
}
```


Links to input drvs

Links to input sources

O.S

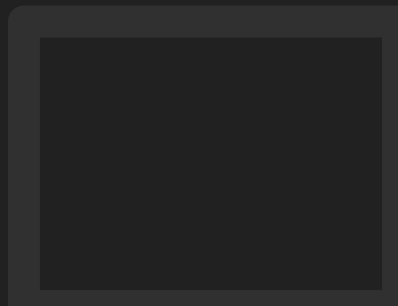
New Feature #6:

(Derivation, Output) → Hash
Trust map on IPFS

Actually still need to do this one

Putting it all together

Finally!



Putting it all together

- All data content addressed
  - Sources, build artifacts, build plans (drvs)
- Same content address used end-to-end
- All data obtained trustless
- Only metadata is (Derivation, Output) → Hash
  - OK, that’s inherently trustful

Further resources

- https://github.com/obsidiansystems/ipfs-nix-guide
  - Tutorial (!)
  - Prose guide to changes behind the scenes (more like talk)
- https://github.com/obsidiansystems/nix/tree/ipfs-master
  - Human-tested branch with all features
- https://github.com/obsidiansystems/nix/tree/ipfs-develop
  - development version