FLUX WIKI BOOTSTRAP

HARDWARE INFRASTRUCTURE, NAS ARCHITECTURE, CORPUS MIGRATION, KEEPER SYSTEM, EMBEDDINGS, AUTONOMOUS INGEST PIPELINE, AND THE LONG-TERM VISION FOR FLUX AS A SELF-ORGANIZING PHOTOGRAPHIC ARCHIVE.

flux.dantesisofo.com/wiki/bootstrap/

FLUX_WIKI_v2.0

JUNE 2026

FLUX DOCUMENTATION SYSTEM Layer 4 — INFRASTRUCTURE | bootstrap
flux.dantesisofo.com/wiki/bootstrap/

BOOTSTRAP

FLUX is not photo editing software.

FLUX is a self-organizing, autonomous, verifiable photographic archive system.

The photographer walks. The photographer photographs. Everything else happens automatically.

1. THE PRINCIPLE

The ideal workflow:

```
Shoot photos
→ transfer once
→ everything else happens automatically
```

The long-term vision:

```
Ricoh GR
→ iPhone / SD card reader
→ NAS
→ FLUX node detects new images
→ archive organization
→ metadata extraction
→ embeddings
→ auto-tagging
→ keeper ranking
→ issue generation
→ blockchain verification
→ publishing
```

The photographer becomes:

```
curator,
not administrator
```

2. HARDWARE

2.1 Compute Node

Apple Mac mini M4 Pro

Role: - FLUX brain - automation node - embedding engine - metadata processor - vector database node - keeper ranking system - issue generation engine - autonomous ingest worker

2.2 Archive Storage

Synology NAS — 2x mirrored IronWolf drives

Role: - canonical archive root - centralized corpus - permanent storage - ingest destination - AI dataset storage - metadata persistence

3. STORAGE PHILOSOPHY

```
NAS    = canonical archive
Mac mini = compute node
S3     = public distribution layer
```

The archive must never become dependent on: - one computer - one drive - one interface

The archive must become: - network-native - modular - persistent - decentralized-ready

4. CANONICAL NAS STRUCTURE

```
/FLUX_ARCHIVE/       ← every photograph ever made
/FLUX_SYSTEM/        ← automation scripts, generators, tools
/FLUX_PUBLIC/        ← staged public assets before S3 sync
/FLUX_METADATA/      ← SQLite databases, JSON metadata
/FLUX_EMBEDDINGS/    ← vector databases, embedding indices
/FLUX_ISSUES/        ← generated PDFs, contact sheets, manifests
/FLUX_LOGS/          ← ingest logs, processing logs, error logs
/FLUX_INBOX/         ← watch folder — incoming photographs land here
```

The inbox is the entry point. Everything flows outward from it automatically.

5. MASTER PHOTO ARCHIVE

5.1 Canonical Corpus

```
/FLUX_ARCHIVE/ORIGINALS/
```

Contains: - every photograph ever made - full chronological corpus - historical truth

Structure:

```
/ORIGINALS/
    /2022/
        /2022-01/
            /2022-01-01/
                2022-01-01_13-22-44_DanteSisofo_R0001234.JPG
```

6. FILE NAMING CONVENTION

Canonical format:

```
YYYY-MM-DD_HH-MM-SS_DanteSisofo_OriginalRicohFilename.JPG
```

Example:

```
2023-01-15_13-22-44_DanteSisofo_R0001234.JPG
```

This ensures: - chronological sorting by filename alone - machine readability - long-term consistency - metadata stability independent of filesystem dates

---

7. SSD MIGRATION PLAN

7.1 Current Situation

The SanDisk SSD contains approximately 400,000 loose Ricoh JPEGs. Mostly unorganized. EXIF metadata intact throughout.

This is not a problem.

The timestamps and EXIF data are the archive. The organization can be reconstructed automatically.

7.2 Migration Strategy

The SSD should not be manually organized first.

Correct flow:

```
SSD (source)
→ Mac mini
→ FLUX migration script
→ EXIF extraction
→ canonical folder generation
→ canonical filename assignment
→ duplicate detection (hash)
→ organized corpus on NAS
```

The migration system will:

1. Scan all loose Ricoh JPEGs
2. Extract EXIF timestamps
3. Generate canonical folder hierarchy (`/YEAR/YEAR-MM/YEAR-MM-DD/`)
4. Rename files to canonical format
5. Move files into chronological structure on NAS
6. Detect duplicates via SHA-256 hash
7. Generate manifest per folder
8. Build initial metadata database

---

8. KEEPERS SYSTEM

8.1 Existing Keeper Archive

A keeper archive already exists containing approximately 15,000 selected images, organized like:

```
/ORIGINALS/
    /2023/
        /2023-01/
            /2023-01-01/
```

These images represent: - taste — the photographer's actual selections - training data for future AI systems - visual truth selections made under real conditions

8.2 Two Simultaneous Representations

Keepers must exist in two forms at once.

1. Human browsing layer:

```
/FLUX_ARCHIVE/KEEPERS/
```

Purpose: browsing, projects, books, exports, emotional interaction.

2. Metadata truth layer:

Inside the full corpus, each file carries a flag:

```
keeper = TRUE
keeper_score = 1.0
```

This transforms the full archive into a labeled machine-learning dataset. The keeper archive becomes the positive class. Everything else is context.

8.3 Keeper Matching

The system will automatically compare the keeper archive against the full corpus using:

- EXIF timestamps
- original Ricoh filenames
- SHA-256 hashes
- image similarity as fallback

To determine: *this image in the full corpus is a keeper.*

The AI must learn: - what gets selected - what gets rejected - near misses - recurring patterns - aesthetic thresholds

This is the foundation of the keeper model.

---

9. METADATA DATABASE

A metadata database becomes the intelligence layer of the archive.

Initial implementation: SQLite
Long-term: PostgreSQL if scale demands it

9.1 Metadata Schema (Target)

```
filename:          2023-01-15_13-22-44_DanteSisofo_R0001234.JPG
sha256:            a3f7...
keeper:            TRUE
```

```
keeper_score:      1.0
issue:             FLUX_VOL_019
camera:            Ricoh GR III
focal_length:      18.3mm
aperture:          f/2.8
shutter:           1/500
iso:               400
location:          Philadelphia
geocoded_address:  South Street, Philadelphia, PA
weather:           fog
time_of_day:       morning
tags:              umbrella, silhouette, layered, shadow
embedding_ref:     vector_8821
motif_cluster:     cluster_umbrella_fog
```

The metadata database is the join layer between the archive (files) and the intelligence layer (embeddings, keeper scores, motif clusters).

---

10. EMBEDDINGS LAYER

Embeddings make the archive computationally alive.

Once every photograph has a vector representation, the archive becomes queryable by visual similarity rather than filename or date.

This enables:

```
show all umbrellas in fog
show visual echoes of Rome
show all silhouettes from winter 2022
show images visually similar to this one
show recurring gestures across six years
show evolution of layering over time
```

Embeddings transform the archive from:

```
files
```

into:

```
relationships
```

The archive stops being storage. It becomes memory.

See INTELLIGENCE for the complete embeddings and keeper model documentation.

---

11. KEEPER MODEL

The system should eventually answer:

```
"What would Dante probably keep?"
```

Not:

```
"What is objectively great photography?"
```

This distinction is critical.

The model is not learning universal aesthetics. It is learning: - personal visual taste - selection philosophy - aesthetic thresholds - recurring motifs - sequencing tendencies

The keeper archive is the training dataset. The full corpus is the test set. The model is a mirror.

---

12. AUTONOMOUS INGEST PIPELINE

12.1 Current State (Implemented)

The existing ingest pipeline already handles the public submission → approval → publish flow:

```
Photographer submits 36 JPEGs via browser
→ Generator processes in-browser (EXIF, PDF, derivatives)
→ Submission uploads to FLUX_SUBMISSIONS/ on S3
→ Portal review: approve / reject
→ approve_worker.py uploads to S3 photos/ + thumbs/
→ Appends to stream.json
→ Syncs stream.json to S3 (public immediately)
→ Auto-triggers issue_builder_worker.py at 36 unassigned frames
→ Issue built, deployed, archive updated
```

The auto-generation trigger (`_maybe_trigger_build()`) detects when unassigned public frames reach 36 and spawns `issue_builder_worker.py` as a detached background process. The portal does not need to be running for the issue to build.

12.2 Long-Term Vision (Future)

The full autonomous pipeline extends this to personal ingest:

1. Detect new images in /FLUX_INBOX/
2. Extract EXIF
3. Generate SHA-256 hash
4. Check for duplicates
5. Organize into chronological hierarchy
6. Rename to canonical format
7. Generate thumbnails
8. Reverse geocode locations
9. Detect weather / time-of-day / daylight conditions
10. Generate embeddings
11. Auto-tag subject matter
12. Compare against keeper dataset
13. Generate keeper scores
14. Generate contact sheets
15. Generate issue drafts
16. Generate manifests
17. Generate blockchain proofs
18. Publish automatically

Steps 1–6: implemented in part for public submissions.
Steps 7–18: future.

---

13. VERIFICATION LAYER

The blockchain layer does not store images.

Each image generates a SHA-256 hash. Each issue generates a manifest containing all frame hashes. The manifest hash is then anchored to an external verification layer.

Verification options:

- **Arweave** — permanent decentralized storage
- **IPFS** — distributed content addressing
- **Bitcoin timestamping** — OpenTimestamps
- **Ethereum** — smart contract anchoring

Result:

```
cryptographically verifiable photographic history
```

Meaning authenticity, chronology, provenance, and existence become mathematically provable.

A single photograph can be forged.
An entire chronological issue — with timestamps, GPS data, sequential frame numbers, neighboring images, publication date, and cryptographic hash anchored to a public blockchain — cannot be retroactively falsified.

The issue is the unit of authenticity.

14. OPEN SOURCE GOAL

The eventual goal:

```
Any photographer
can run their own FLUX node
```

What this means: - install FLUX - connect an archive - own your data - train a personal keeper model - generate issues - publish independently - avoid centralized platforms

Without: - Adobe lock-in - Instagram dependency - centralized archival control - cloud subscription fees - algorithmic ranking

FLUX should become open protocol, not proprietary product.

15. PHASED ROADMAP

Phase 1 — Infrastructure

1. Setup Synology NAS
2. Configure mirrored IronWolf drives
3. Establish canonical folder structure
4. Mount NAS to Mac mini
5. Establish /FLUX_INBOX/ watch folder

Phase 2 — Corpus Migration

6. Connect SanDisk SSD
7. Run EXIF extraction pass
8. Generate canonical folder hierarchy
9. Rename all files to canonical format
10. Migrate corpus onto NAS
11. Build master corpus: ~400,000 frames organized

Phase 3 — Keeper Integration

12. Scan existing keeper archive
13. Match keepers against full corpus
14. Flag keepers in metadata
15. Build labeled keeper dataset (~15,000 positives)

Phase 4 — Metadata Layer

16. Initialize SQLite database
17. Generate SHA-256 hashes for all frames
18. Generate manifests per session and issue
19. Generate thumbnails for full corpus
20. Build metadata schema

Phase 5 — Embeddings

21. Generate visual embeddings for full corpus
22. Build vector database
23. Build similarity search
24. Build motif clustering
25. Build semantic archive search interface

Phase 6 — Keeper Intelligence

26. Train keeper ranking model on labeled dataset
27. Build keeper scoring for new ingest
28. Build auto-selection experiments
29. Build automated issue sequencing drafts

Phase 7 — Autonomous FLUX

30. Build watch-folder ingest daemon
31. Build fully autonomous issue generation
32. Build automatic publishing
33. Build blockchain verification pipeline
34. Build decentralized archival persistence layer

---

16. FINAL PRINCIPLE

The goal is not: - photo storage - photo editing - portfolio management

The goal is:

a living,
self-organizing,

autonomous,
verifiable,
decentralized
photographic memory system

The archive is not a product. It is infrastructure for a life of continuous seeing.

SEE ALSO

Document	Layer	Relationship
INTELLIGENCE	Layer 5 — Intelligence	The keeper model and embeddings layer that Phase 5–6 build
PHYSICAL	Layer 6 — Physical	The autonomous print pipeline that completes the Bootstrap vision
PRESERVATION	Layer 7 — Preservation	The blockchain verification layer (Phase 7)
ROADMAP	Layer 8 — Roadmap	The phased development timeline
REMOTE ACCESS	Layer 4 — Infrastructure	Tailscale access to the Portal that runs on this hardware

FLUX_WIKI_v1.1 — flux.dantesisofo.com/wiki/bootstrap/