Migration Decision Worksheet

A four-step worksheet for migrating a single-user workflow off Dropbox.

Ronald 'Ryy' G. Thomas

2026-05-07

Table of contents

2026-05-07 16:11 PDT

This worksheet is the companion to the blog post 'Migrating Off Dropbox: Beyond Dotfiles'. It walks through four steps: inventory, mechanism selection, migration order, and verification. Fill it in for your own setup; the result is a concrete migration plan whose moves are small and auditable.

The worksheet is layer-agnostic in the sense that it does not assume any particular sync mechanism. The choices it surfaces depend on the reader's tolerance for new tools, willingness to host their own infrastructure, and constraints on cost and bandwidth.

Step 1: Inventory

List every sync-relevant artefact in `$HOME` and `~/Dropbox/` (or `~/Library/CloudStorage/Dropbox/`). Classify each into exactly one of three layers.

Layer 1: Configuration (dotfiles)

These are files whose content is small, changes infrequently, is nearly always edited from a single machine, and benefits from version history.

- ☐ `~/.zshrc`, `~/.zshenv`, and any other shell rc files
- ☐ `~/.gitconfig`, `~/.gitignore_global`
- ☐ `~/.config/<app>/` directories that hold text config
- ☐ Editor configuration (`~/.vimrc`, `~/.config/nvim/`, etc.)
- ☐ Tool config that is not secret (`~/.npmrc`, `~/.lintr`, etc.)
- ☐ launchd plists (macOS) or systemd user units (Linux)

These belong in the dotfiles repository established by post 24. If they are not there yet, the rest of the worksheet should wait; post 24 is the prerequisite.

Layer 2: Project content

These are directories holding research repositories, datasets, drafts, archives, and shared resources.

- ☐ `~/Dropbox/prj/` (research and software repositories)
- ☐ `~/Dropbox/docs/` (notes, drafts, references)
- ☐ `~/Dropbox/sbx/` (sandboxes and one-off scripts)
- ☐ `~/Dropbox/work/` (employer-related material)
- ☐ `~/Dropbox/shr/` (shared assets, templates, bibliographies)
- ☐ `~/Dropbox/rgt_archive/` (archived material)

For each directory above, decide whether it splits further into 'code' (wants git) and 'data' (wants something else). Many readers find that `prj/` is mostly code, `docs/` is mixed, and `shr/` is mostly data.

Layer 3: Append-only history

These are files written by tools that append rather than overwrite. Cloud sync handles them poorly.

- ☐ `~/.zsh_history` (shell command history)
- ☐ `~/.viminfo` (vim per-session state)
- ☐ `~/.claude/history.jsonl` and other AI-tool histories
- ☐ Editor undo files in `~/.local/state/<editor>/undo/`
- ☐ Terminal history files (`.bash_history`, etc.)

If you are on a single machine and have never seen a 'conflicted copy' file, this layer's payoff is small and you may skip it. If you have, this is where most of the friction lives.

Items NOT in any layer

Some files at `$HOME` are neither configuration nor project content nor history: caches, runtime state from individual applications, secrets. These are explicitly out of scope for this worksheet.

- `~/.cache/`, `~/Library/Caches/` — regenerable; do not sync
- `~/.aws/`, `~/.ssh/`, `~/.gnupg/`, `~/.password-store/` — secrets; hand-installed per machine, never in any sync mechanism
- `~/.docker/`, `~/.npm/`, `~/.lmstudio/` — application state; treat as machine-local

Step 2: Mechanism Selection

For each layer, pick a sync mechanism. The choice depends on what the layer's items actually need.

Layer 1: Dotfiles

Default: **git remote (private repo)**. Use the dotfiles repository established by post 24. No alternative is competitive for this layer.

Layer 2: Project content

Decide for each subcategory below. Most readers will end up with two mechanisms (one for code, one for data).

Code (text-heavy, version-history-valuable)

- ☐ **Per-project git remote.** GitHub, GitLab, or Codeberg. Free for public; private repos are typically free up to a collaborator limit.
- ☐ **Self-hosted git (Gitea, Forgejo).** Requires a server (NAS, VPS, home machine). No third-party quotas.
- ☐ **Git plus a sync layer for working trees.** For projects that exist as working trees on multiple machines but are not yet ready for a remote. Syncthing handles this cleanly.

Data (binary, large, often immutable)

- ☐ **Syncthing (peer-to-peer).** No third-party hosting; LAN sync is fast; supports excludes; conflict resolution is simpler than Dropbox's. Best for two to four machines that are usually on the same network.
- ☐ **rsync over SSH to a NAS or VPS.** Bandwidth-efficient, scriptable. One-way push by default; bidirectional needs a tool such as `unison`. Best when one machine is canonical and others are clones.
- ☐ **Cloud object storage (S3, GCS, B2).** With `rclone`. Inexpensive at scale; archival-friendly with cold storage tiers. Best for data that is read often but rarely written.

- ☐ **iCloud Drive (macOS) or Google Drive.** Native client; familiar UX. Same whole-file model as Dropbox; the conflict-copy class of failure persists.
- ☐ **Stay on Dropbox.** Acceptable if the only Dropbox issue was the dotfiles layer. Skip the move; just point `$PRJ_ROOT` at the Dropbox path.

Per-project decision

For each project directory listed in Step 1, write down:

- Code mechanism: _____
- Data mechanism: _____
- Excludes (`.gitignore`, `.stignore`, etc.): _____

Layer 3: Append-only history

- ☐ **Per-machine state.** Do not sync at all. Simplest; recommended unless one of the histories below is genuinely load-bearing.
- ☐ **Single-machine writes.** Designate one canonical machine; others write to local files that are not synced.
- ☐ **Append-aware sync (atuin for shell history).** Solves the shell-history case cleanly. No equivalent maintained tool for AI-tool histories; hand-rolled mergers are possible but out of scope.

Step 3: Migration Order

Sequence the moves so the backup pipeline never points at a stale source path and the workflow is functional at every intermediate step.

Step	Action	Why now
1	Confirm post 24's dotfiles repo works on a clean account or VM	Foundation for everything else
2	Add `PRJ_ROOT` env var to `~/.zshenv`; resolve to current Dropbox path	Indirection lets the rest of the moves be one-line edits
3	Replace literal `~/Dropbox/prj` references in scripts and configs with `$PRJ_ROOT`	Same: indirection now, behaviour later
4	Move one pilot project to `~/prj/<project>/` (real path, outside cloud)	Verifies the new mechanism in isolation
5	Push pilot to its chosen remote (per-project git, Syncthing folder, etc.)	Confirms the chosen mechanism end-to-end

Step	Action	Why now
6	Migrate the rest of project content, in batches	Bounded blast radius; backup pipeline never goes stale
7	Repoint `$PRJ_ROOT` to the new path	The single switchover moment
8	Update backup scripts (`backup-gdrive`, `backup-icloud`, etc.) to source from `$PRJ_ROOT`	Backup pipeline now follows the workflow
9	Apply the chosen Layer 3 strategy (per-machine, single-writer, or append-aware)	History is the last move because it has the least leverage
10	Verify on a clean account or VM	The end-to-end test that catches every assumption
11	Optional: uninstall Dropbox	Most readers skip this; Dropbox staying installed is fine

The key invariant: at every step, the workflow on the current laptop continues to function. If you stop after step 4, you have a pilot. If you stop after step 8, you have full project-content independence and can defer history decisions indefinitely.

Step 4: Verification

The migration is complete when all four checks pass.

- ☐ **Fresh-account bootstrap.** On a new user account or VM, run `git clone … ~/dotfiles && ./install.sh`, then clone the project remotes. Confirm the workflow comes up without Dropbox being installed.
- ☐ **Backup pipeline still firing.** Confirm that `backup-gdrive` and `backup-icloud` (or your equivalents) run on schedule and produce non-empty output. Most failures are silent; explicit log inspection is the only reliable check.
- ☐ **No conflict-copy regrowth.** A week after migration, grep for 'conflicted copy' across `$HOME` and `$PRJ_ROOT`. The result should be empty.
- ☐ **Audit and document.** Record the chosen mechanisms in the dotfiles repo's README so future-you can audit and so a collaborator (or LLM) can pick up the context.

Notes

- This worksheet does not generate scripts. The reader still has to write the rclone or rsync command for their setup. The blog post body has worked examples for the most common cases.
- The mechanism table is current as of 2026-05; new entrants (Filen, S3-backed clients, etc.) may shift the optimal choice.

- The worksheet is layer-agnostic but assumes a single user. Team setups have an additional concern (the shared folder) that this worksheet does not address.

Rendered on 2026-05-07 at 16:11 PDT. Source: ~/Dropbox/prj/qblog/posts/64-migrating-off-dropbox/migrating-off-dropbox/docs/migration-decision-worksheet.qmd