PH7X SYSTEMS®

0.0.0 · CHAPTER 8

Verify what actually moved

SharePoint Migration Field Guide · Chapter 8

pH7x Systems

2026-08-18

8. Verify what actually moved

> A migration report is the producer's account of its own work.

The tool has run. The progress bar reached the end, the summary says completed, and somewhere a spreadsheet of green rows is waiting to be attached to an email. This chapter is about the question that spreadsheet cannot answer:

> **What actually arrived, and what would prove it to someone who does not trust the tool?**

A migration report is not independent evidence

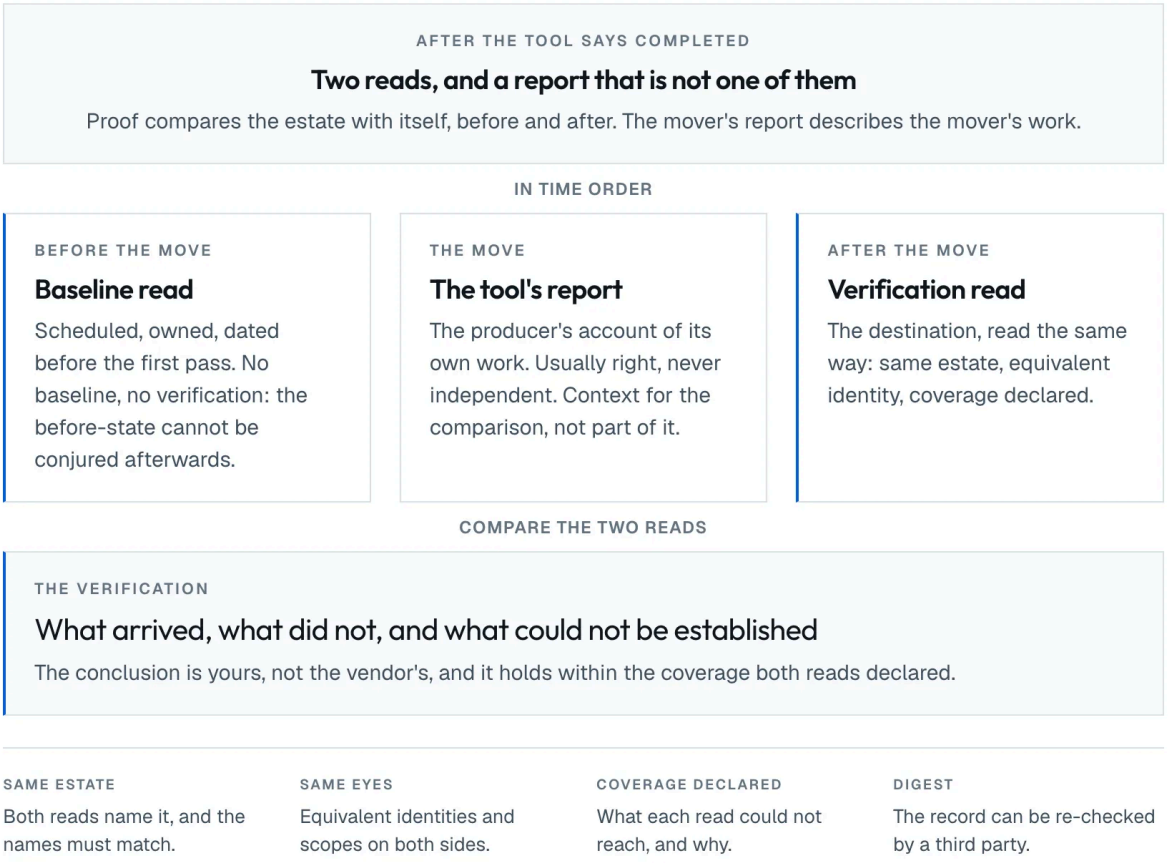
Every migration tool produces a report, and every one of those reports has the same structural problem: it is written by the party whose work it describes. A report that says "4,812 files migrated" is the mover telling you what the mover believes it did. It may be right. It usually is. But when a file is missing two years from now, in front of an auditor or a court, "the tool said completed" is not evidence; it is a quotation.

The proof has a different shape, and it needs two documents the tool did not write: a read of the source taken **before** the move, and a read of the destination taken **after** it. Compare the two, and the conclusion belongs to you, not to the vendor. That is the whole method of this chapter, and everything below is the discipline of doing it honestly.

The baseline is taken before, or never

The single most common verification failure is not a technical one. It is a calendar one: nobody reads the source until after the migration, and by then the source is being decommissioned, is read-only, is half-deleted, or is simply no longer trusted. A verification attempted at sign-off, against a source that is already gone, has nothing to compare and quietly degrades into an inventory of the destination.

So the baseline read is a scheduled project task with an owner and a date, and the date is before the first migration pass. Chapter 9 will refine this, because the source keeps changing after the baseline; for now the rule is absolute: **no baseline, no verification**. There is no tool, ours or anyone's, that can conjure the before-state after the fact.



Say what was read, and what could not be

A read is only evidence if it declares its own boundaries. Two declarations matter:

- **The estate.** What was this read of? A named site collection, a set of libraries, a file share path. Written as the operator names it, so that the baseline and the verification can be checked to be reads of the *same* estate.
- **The coverage.** What could this read *not* reach, and why? A folder the account was refused, a path too long to enumerate, a library skipped by decision. A verification that cannot say what it failed to read is indistinguishable from one that read everything, and the two are opposites.

Coverage is the most important and least recorded part of a read. If the baseline covered 96 sites and the verification covered 94, the two missing sites are not "fine"; they are two places where the answer is

we do not know

, and the record must say so.

Eight dimensions, and what each one proves

"Did it move?" is not one question. It is at least eight, and a match on one proves nothing about the others.

DID IT MOVE?

Eight questions, not one

A match on one dimension proves nothing about the others, and an unreadable item is never a failure.

DIMENSION	A MATCH PROVES	IT DOES NOT PROVE	IF UNREADABLE
Presence	The item exists on both sides	Anything about its contents	unknown, with the side
Count	Nothing dropped in bulk	Which items are the same	unknown
Size	The bytes weigh the same	That they are the same bytes	unknown
Content	Same bytes, if compared by digest	**Anything, if compared by size alone**	unknown
Authorship	Created-by and modified-by survived	That versions behind them did	unknown
Versions	History depth survived	What each version contains	unknown, or a stated limit
Permissions	Access rules arrived, translated	That they mean what they meant	unknown
Sharing links	Sharing carried, or consciously reset	Who used the old links	unknown

DIMENSION	A MATCH PROVES	A MATCH DOES NOT PROVE
Presence	The item exists on both sides	Anything about what it contains
Count	Nothing was silently dropped in bulk	Which items are the same items
Size	The bytes weigh the same	That they are the same bytes
Content	The bytes are the same, if compared by digest	Anything, if compared by size alone
Authorship	Created-by and modified-by survived	That versions behind them survived
Versions	The history depth survived	What each version contains
Permissions	Access rules arrived	That they mean what they meant
Sharing links	Sharing state was carried or consciously reset	Who used the old links

Three of these deserve their own warnings, because the platform's documented behaviour makes naive comparison wrong.

Versions. What survives depends on the path taken. From a file share, previous versions do not migrate at all: Microsoft's own FastTrack migration table lists ownership history and previous versions as not migrated (Data Migration, checked on 18 August 2026). From SharePoint Server, the SharePoint Migration Tool migrates checked-in versions but not the checked-out version of a file (SPMT release notes, checked on 18 August 2026). So a version-count mismatch is not automatically a loss: it may be the documented behaviour of the road you chose. The verification record should distinguish a limit of the path from a defect of the run, because only the second one is actionable.

Permissions. These are translated, not copied. On a file-share migration, Write becomes Contribute, explicit Deny entries are not saved at all, and from SharePoint Server only unique permissions migrate while inherited ones are recreated by inheritance at the destination (File and folder permissions when migrating, checked on 18 August 2026). Comparing raw ACL lists across that translation produces noise; the honest comparison is of *effective* access for named principals, and any Deny entry in the source is a finding by definition, because the destination cannot express it.

Sharing links. Microsoft's guidance for migrating shared content is explicit that anonymous sharing links should not be migrated at all: it is not possible to know who held the old link, and users should create new ones deliberately (Migrating shared files and folders, checked on 18 August 2026). The same page documents that sharing is two facts, a permission and a shared-with reference, and both must exist for the item to appear as shared. A verification that only counts links misses half the mechanism.

Content, and the digest that can lie honestly

Size is cheap to compare and settles almost nothing: two different documents can weigh the same. If the verification claims anything about *content*, it must compare digests, and the record must say which method it used. SharePoint and OneDrive expose a content hash for files, computed with the QuickXorHash algorithm (QuickXorHash code snippets, checked on 18 August 2026); compute the same digest over the source bytes and you have a real content comparison without downloading the destination.

One honest complication: for Office formats, the service can rewrite internal package parts on ingestion, a behaviour called property promotion, so a digest computed over the destination file can differ from the source even when nothing a human would call content was lost. A digest mismatch on a `.docx` is therefore a finding to *explain*, not automatically a loss. This is exactly why the record needs an outcome vocabulary richer than pass and fail.

`missing` is not `unknown`

Here is the sentence this chapter exists for: **not observed is not the same as missing.**

When the verification read cannot reach an item, the truthful outcome is *unknown*, with the side that failed and the reason: permission denied, path too long, item checked out. When the read reached the place and the item is not there, the outcome is *missing*. Most reports collapse these into one, and the collapse destroys the document, because the response to each is different: an `unknown` is a coverage problem you fix by reading again with better access; a `missing` is a loss you fix by migrating again. A report that cannot tell them apart sends people to re-migrate content that was never lost, and to sign off content that was never checked.

The same discipline applies to what the source could never expose. A file share has no version history to read, no shared-with references, no created-by beyond the filesystem owner. Those dimensions are not

failures

of the verification; they are limits of the source, and the record should declare them as dimensions not compared, with the reason, so a reader does not infer from silence that they were checked.

The same eyes on both sides

Both reads must be taken by identities with equivalent access, and the record must say who read each side. If the baseline was read by a tenant administrator and the verification by a departmental account, every item the second account cannot see becomes an `unknown` at best and a false `missing` at worst: a permission difference wearing the clothes of a loss. In our experience this is the most common source of false alarms in verification, and it is entirely preventable: record the identity kind, the account, and the scopes with each read, and refuse to compare presence across reads taken by different eyes.

The record

The output of this chapter is not a feeling of confidence. It is a document, and the document has a defined shape: the two reads it compared, named by identity and digest; the dimensions it compared and the ones it did not, each with a reason; and one finding per item and dimension that did not pass silently. This site publishes the full contract for such a record, including its schema, at migration-verification/1.0.0; the excerpt below shows the parts that carry the method, produced by the open-source engine this site documents:

```
{
  "$schema": ".../migration-verification/1.0.0",
  "baseline":     { "read_id": "baseline-001",     "taken_at": "2026-07-04T06:10:22Z",
                    "estate": "contoso-projects", "canonical_hash": "9f2c…",
                    "read_by": { "kind": "application", "scopes": ["Files.Read.All"] } },
  "verification": { "read_id": "verification-001", "taken_at": "2026-08-11T05:58:41Z",
                    "estate": "contoso-projects", "canonical_hash": "b41a…",
                    "read_by": { "kind": "application", "scopes": ["Files.Read.All"] } },
  "dimensions": [
    { "name": "presence", "state": "compared" },
    { "name": "content",  "state": "compared", "method": "digest" },
    { "name": "versions", "state": "not-compared",
      "reason": "file-share source carries no version history",
      "limit": "not-supported" }
  ],
  "findings": [
    { "item": "/Shared Documents/Archive/2019.xlsx",
      "dimension": "presence", "outcome": "unknown",
      "side": "verification", "state": "permission-denied" }
  ]
}
```

Whether you produce this with our tooling or with your own scripts matters less than the properties: taken before and after, by equivalent identities, with coverage declared, with `unknown` kept apart from `missing`, and with a digest so a third party can check the document itself was not edited after the fact.

What this still does not prove

A clean verification record proves that what the baseline saw is present, intact and equivalently governed at the destination, within the declared coverage. It does not prove the migration was *complete*: anything created in the source after the baseline is invisible to it, which is the entire subject of Chapter 9. It does not prove the destination *works*: navigation, customisations and integrations are functional questions, not inventory ones. And it does not prove the content is *correct*, only that it is the same content the source held. We recommend saying all three of these out loud in the sign-off meeting, because the alternative is that "verified" quietly inflates into a promise no inventory comparison can keep.

Out of scope

This chapter verifies a move that has already happened against a baseline that already existed. Scheduling the passes, freezing the source, the final delta and the cutover decision are Chapter 9. Choosing what

should

move at all was Part I, and the measurement of the source estate it relies on is the rest of Part II.

Before you continue

- [ ] The baseline read is a scheduled task with an owner, dated before the first migration pass
- [ ] Both reads declare their estate and their coverage, including what they could not reach
- [ ] The dimensions compared are written down, and every dimension not compared carries a reason
- [ ] Content claims rest on digests, not sizes, and the digest method is named in the record
- [ ] `unknown` findings are separated from `missing`, with the side and the reason on each
- [ ] Both reads were taken by identities with equivalent access, and the record names them
- [ ] The verification record itself carries a digest a third party can recompute