COMPUTER FORENSICS LAB

London digital forensics & e-discovery · Established 2007

GitHub, GitLab and Source Code as Evidence

Commits, Clones and the Development History That Dates Every Line

Software disputes: IP theft, copied codebases, departing developers, failed projects, licence breaches: are decided in version control. Git records who committed which lines when, across every branch and fork; the platforms around it (GitHub, GitLab, Bitbucket) add pull requests, reviews, issues, access logs and clone records. Read properly, a repository is a dated, attributed manuscript of the software's entire life. Read naively, it misleads: commit dates can be forged, authorship is a settable field, and history can be rewritten. This guide gives UK lawyers the working anatomy of git and platform evidence, the authentication techniques that separate real history from rewritten history, the clone-and-fork trail in exfiltration cases, code-similarity analysis in copying disputes, and the collection methods that preserve repositories evidentially.

⏱ Estimated reading time: 25 min read

§ ABOUT THE AUTHOR

PREPARED BY COMPUTER FORENSICS LAB E-DISCOVERY TEAM

Computer Forensics Lab is a London-based digital forensics and electronic disclosure practice established in 2007. Source-code matters feature across its IP, employment and software-dispute casework: repository preservation and authentication, commit-history analysis, clone and access-log investigation on GitHub and GitLab estates, and code-similarity examination under CPR Part 35. Its eDiscovery arm, **e-discovery.uk**, hosts code productions beside the correspondence estate.

ESTABLISHED 2007 · LONDON | ISO 17025-ALIGNED PROCEDURES | REPOSITORY & COMMIT FORENSICS | CODE-SIMILARITY ANALYSIS | CPR PART 35 EXPERT REPORTS | FULL CHAIN-OF-CUSTODY DOCUMENTATION

§ CONTENTS

In this guide

§ 0 1 · ORIENTATION

Executive summary

THE HEADLINE POINT: GIT DATES EVERY LINE AND NAMES EVERY AUTHOR, BUT BOTH FIELDS ARE SETTABLE: PLATFORM-SIDE RECORDS ARE THE ANCHOR THAT MAKES REPOSITORY HISTORY TRUSTWORTHY

A git repository is the strongest development chronology available: every commit records author, timestamp, message and the exact changes, chained by cryptographic hashes so that altering any past commit changes every hash after it. But the local fields are user-set: commit dates can be forged and authorship misattributed by anyone at a keyboard: so evidential weight comes from corroboration: the platform's server-side records (push times, pull requests, CI runs, webhook and audit logs) which the committer does not control, cross-repository hash comparison, and the wider estate (emails discussing the code, deploy records). Exfiltration cases run on the platform's clone, fork and access logs; copying disputes run on similarity analysis anchored to provenance. Collection preserves full repositories (all branches, tags and metadata) plus the platform layer, hashed and documented.

- **The hash chain is the integrity spine:** rewritten history changes hashes: divergence between copies of "the same" repository is itself the finding.
- **Local fields are assertions, server records are facts:** commit dates and authors verify against push times, PRs and CI logs before being relied on.
- **Clone and access logs decide exfiltration cases:** platform-side, retention-bound, frozen in week one.
- **Copying is proved by similarity plus provenance:** shared oddities, comments and mistakes, tied to a dated ancestor.
- **Collect repositories whole:** every branch, tag, and the platform metadata: a zip of the current files is not the evidence.

§ 0 2 · FIRST PRINCIPLES

The problem in plain English: the manuscript that dates itself

Imagine a manuscript where every editing session was saved as a numbered, dated page listing exactly which words changed and who changed them, and where each page carries a seal computed from all the pages before it, so that tampering with page 40 visibly breaks the seals on pages 41 onward. That is a git history. It answers litigation's development questions: when was this feature written, who wrote this function, did this code exist before the contract date, was the fix made before or after the incident: with line-level precision.

The caveat that shapes everything: the dates and names on each page were written by the person saving it. Honest workflows make them reliable; a motivated party can backdate and misattribute freely on their own machine. The discipline of this guide is therefore two-layered: read the repository for its story, and authenticate the story against records the storyteller does not control: the platform's server-side logs, other copies of the repository, and the world outside the repo.

§ 0 3 · THE REPOSITORY

Git anatomy: commits, branches, hashes and what they prove

- **Commits:** the unit of history: author and committer fields (name, email, each with its own timestamp), message, parent references and the content snapshot: author and committer can differ, and the difference (cherry-picks, rebases, patches applied by others) is often meaningful.
- **Hashes:** each commit's identifier is computed from its content, metadata and parents: identical hashes across repositories prove identical history; a hash present in both parties' repos proves shared ancestry to the second.
- **Branches and merges:** parallel development lines and their joins: the topology shows how work actually flowed, and orphaned or deleted branches may survive in reflogs and clones.
- **Tags and releases:** named markers on specific commits: the "what was version 2.1" question answered exactly.
- **The working tree vs the history:** the current files are one snapshot; the evidence is the whole object database: which is why collection clones with full history rather than zipping a folder.

§ 0 4 · THE PLATFORM LAYER

Platform evidence: pull requests, reviews, issues and logs

- **Pull/merge requests:** proposed changes with server-side creation and merge timestamps, reviewer comments and approvals: contemporaneous, attributed discussion of the very code in dispute, and a date anchor commits cannot forge.
- **Issues and project boards:** the bug reported, when, by whom, and what was said: failed-project and defect litigation's narrative layer.
- **CI/CD records:** build and test runs triggered by pushes, with server timestamps: proving code existed and behaved (or failed) at dated moments: deployment logs extending the chain into production.
- **Audit and access logs:** organisation-level events: logins, permission changes, repository creation and deletion, and (tier-dependent) clone and download records: retention-bound and exported early.
- **Protected-branch and signing policies:** where enforced (required reviews, signed commits), the workflow's guarantees strengthen the history's reliability: the configuration itself is evidence of what could and could not have happened silently.

§ 0 5 · TRUST, VERIFIED

Authenticating history: forged dates and rewritten pasts

- **Date-forgery checks:** commit timestamps tested against push times (server-side), CI runs, PR records and the correspondence estate: a commit "from 2023" first pushed in 2026 carries its own cross-examination.
- **Rewrite detection:** force-pushes and history rewrites leave traces: platform events, diverging hashes against other clones, reflog remnants, and dependency or toolchain anachronisms inside the code (a 2023 commit importing a library released in 2025 dates itself).
- **Author verification:** the author field checked against the platform account that pushed, signing keys where used, and the human's working pattern: the attribution ladder of this series, code edition.
- **Clone triangulation:** every developer's machine, every fork and every backup holds an independent copy of history as it stood when taken: divergence maps exactly what was rewritten, and when the copies stop agreeing.
- **The honest bound:** a lone repository with no platform history and no counterpart clones authenticates weakly: findings state the corroboration reached, per the standing fairness rule.

§ 0 6 · TAKEN

Exfiltration: clones, forks and the departing developer

- **The clone event:** one command copies the entire repository with its whole history: platform clone/download logs (tier-dependent) date it; device artefacts (guide 41) evidence the local copy; personal-account forks leave server-side records.
- **Token and key trails:** personal access tokens and SSH keys: issuance, use and the IPs behind them: post-resignation access frequently rides an unrevoked token, and the token log is the proof.
- **The reappearance:** the taken code surfacing in the new employer's product: proved by §7's similarity-plus-provenance method, with shared hashes as the gold standard where the thief was lazy enough to keep the history.
- **Baseline fairness:** developers clone legitimately every day: the finding is pattern: timing against notice, scope beyond assigned work, personal-account destinations: pleaded to its rung.
- **Immediate response:** tokens and keys revoked, audit and access logs exported before rotation, and the §11 preservation letter out: the platform layer's half-life sets the clock.

§ 0 7 · COPIED

Copying disputes: similarity analysis and provenance

- **Similarity with judgement:** matching code is the start, not the finding: independent implementation, common libraries, generated boilerplate and AI-assisted output all produce innocent resemblance: the analysis separates the explicable from the damning.
- **The damning class:** shared comments and typos, identical variable-naming quirks, copied dead code and bugs, matching internal structure with no functional cause: fingerprints of the ancestor, not the problem.
- **Provenance anchoring:** the claimant's dated history proving priority: which lines existed, when, per the authenticated ladder: the copying case is chronology plus fingerprints.
- **Licence-breach variants:** open-source components and their obligations traced through dependency manifests and vendored code: the compliance question answered from the repository itself.
- **Protocolled comparison:** where both sides' code is confidential, comparison runs under an expert protocol (designated examiners, secure environments, findings reported without wholesale disclosure): guide 68's intermediary logic between competitors.

§ 0 8 · THE WIDER MAP

Source architecture: where else the evidence lives

Per this series' source-architecture discipline: the platform repository is one node in a distributed estate: git's design copies full history to every participant. The evidence also lives in: every developer's local clone (independent history snapshots, plus reflogs recording even rewritten states); forks under other accounts; CI/CD systems holding checked-out copies, build artefacts and logs; deployment targets and container registries holding what actually ran, dated; package registries holding published versions; backup systems and old laptops holding period clones; the issue-tracker and correspondence estate discussing the code; and counterpart repositories (the alleged copier's own history). When the primary repository has been rewritten or deleted, the clones and forks are the record: history that was distributed cannot be centrally unmade.

EVIDENCE	PLATFORM REPO	LOCAL CLONES / FORKS	CI/CD & DEPLOYS	REGISTRIES & ARTEFACTS	BACKUPS / OLD DEVICES	DELETED / RECOVERABLE
Commit history	Y: canonical, rewritable	Y: independent snapshots + reflogs	Checked-out states	Published snapshots	Period copies	Distributed: rewrites diverge, clones remember
Push / access events	Y: server logs, retention-bound	n/a	Trigger records	Publish logs	n/a	Exported early or lost
PRs / reviews / issues	Y: server-side	n/a	Linked runs	n/a	API exports sometimes	Platform retention governs
What ran in production	Tags assert	n/a	Y: deploy logs	Y: images/ packages, dated	Y	Artefact stores persist
Exfiltration events	Clone/token logs	The copy itself	n/a	n/a	Guide 41's device layer	Chain per guides 41/69

Fallback strategy in one line: when the canonical repository is rewritten or gone, reconstruct from clones, forks, CI checkouts and published artefacts: and let the divergence date the tampering.

§ 0 9 · IN THE WILD

Worked examples

EXAMPLE 1 · THE COMMIT FROM THE FUTURE'S PAST

An IP defendant produced a repository showing its disputed algorithm committed in March 2023, ten months before the claimant's release: independent prior creation, pleaded with dates. Authentication unpicked it: the platform's records showed the entire repository first pushed in June 2025, in one batch; the "2023" commits imported a mathematics library at a version released in September 2024; and the CI history began, like everything server-side, in 2025. The local dates were typed; the server dates were earned. Under the schedule of anachronisms the prior-creation defence was abandoned, and the court's costs language about "manufactured chronology" followed the defendant into the damages hearing.

EXAMPLE 2 · THE TOKEN THAT OUTLIVED THE JOB

A fintech's lead developer resigned; three weeks later a rival demonstrated a suspiciously mature product. The platform's logs held the sequence: a personal access token, created two days before resignation, used to clone all fourteen private repositories at 05:50 on his final morning from a residential IP; the token then used twice more, post-employment, to pull the latest commits: unrevoked, still valid, still logged. Device forensics found the clones' object databases on his personal NAS: with commit hashes identical to the employer's, shared ancestry proved to the mathematical standard of §3. The springboard injunction cited the 05:50 clone; the post-employment pulls converted a theft case into an ongoing-access case; and token-revocation-on-exit entered the client's leaver checklist permanently.

EXAMPLE 3 · THE TYPO THAT TRAVELLED

Two competing logistics platforms shared no code, said the defendant: parallel engineering, common problem, similar solutions. The similarity examination found the fingerprint class: a comment reading "TODO: fix this horible hack before realease" identical in both codebases, characters and misspellings intact; the same dead debugging function, never called, in both; and matching variable quirks through the routing module. Provenance sealed it: the claimant's authenticated history showed the comment written in 2022 by a named developer: who had joined the defendant in 2023. The defendant's repo, created three months after his arrival, had no history before its first all-at-once commit. The inference ran one way, and the case settled at mediation with the comment blown up on an exhibit board.

§ 10 · WHERE IT GOES WRONG

Common mistakes and technical limitations

Common mistakes

- **Zips instead of clones:** current files collected, the object database: the actual evidence: left behind.
- **Commit dates taken at face value:** Example 1's trap: local assertions relied on without server corroboration.
- **Platform logs left to rotate:** clone, token and audit records: the exfiltration case: lost to retention while letters were exchanged.
- **Tokens and keys unrevoked:** Example 2's post-employment pulls: the breach continuing through the investigation.
- **Similarity without provenance:** matching code pleaded without the dated ancestor: the innocent-resemblance defence left open.
- **Single-copy reliance:** one party's repository treated as the history, no clone triangulation sought.
- **Confidentiality handled late:** competitor code exchanged without the §7 protocol, poisoning the comparison's admissibility and the parties' tempers alike.

Technical limitations

- **Local metadata is settable:** dates and authors are claims until corroborated: the standing caveat behind every finding.
- **Platform log depth is tiered:** clone and audit records vary by plan and configuration: capability audited before promises.
- **Rewrites can be clean:** a careful rewrite with no surviving clones may be undetectable from the repo alone: the honest bound stated.
- **Similarity has innocent causes:** frameworks, generators and AI assistance produce resemblance: findings rest on the fingerprint class, not volume.
- **Reflogs are local and expiring:** the rewrite-remembering layer lives on machines and ages out: device preservation is part of repository preservation.

§ 11 · INTERROGATORIES & DRAFTING AIDS

Questions to ask · Suggested wording

Ask your client

- Which platform, plan and audit configuration: and are the clone, token and audit logs exported before rotation?
- For leavers: are all tokens, keys and sessions revoked, and which repositories could they reach?
- Where are the independent copies: developer machines, CI, backups: that triangulate the history?

Ask your opponent

- Please preserve and produce the repositories in full: all branches, tags and complete commit history in native git form: together with the hosting platform's records: pull requests, reviews, issues, CI logs, and organisation audit, access and clone logs for [period].
- Please state when each repository was created on its current platform, identify any migration or import, and confirm whether history has been rewritten, force-pushed or filtered, producing the platform events recording any such acts.
- Please identify all persons and tokens with access to [repositories] during [period], and any clones, forks, downloads or exports, producing the corresponding logs.

Ask your eDiscovery / forensic provider

- Does the server-side record corroborate the commit chronology relied on: and where are the anachronisms?
- What do the clone, token and access logs show against the departure timeline?
- Does the similarity rise to the fingerprint class, and does our provenance ladder hold?

SUGGESTED WORDING · SOURCE-CODE LIMB FOR THE PRESERVATION LETTER

> "Your client must immediately preserve all source code repositories relating to the issues, in native version-control form with complete history: all commits, branches, tags and metadata: which must not be rewritten, force-pushed, filtered, migrated or deleted. Your client must further preserve and export forthwith the hosting platform's server-side records, including pull and merge requests, code reviews, issues, CI/CD logs and all organisation audit, access, clone and token logs, whose rotation must be suspended so far as within its control; all developer local clones and reflogs on devices within its control; and all build artefacts, published packages and deployment records evidencing the code as run. Please confirm within [3] days, identifying the platform, plan and audit configuration."

§ 12 · QUICK CONTROL

Checklist and red flags · When to involve a digital forensic expert

The source-code checklist

- ☐ Repositories collected as full clones with all branches, tags and metadata, hashed
- ☐ Platform layer exported: PRs, reviews, issues, CI, audit/access/clone logs
- ☐ Log retention checked; rotating records exported week one
- ☐ Tokens, keys and sessions revoked in leaver matters
- ☐ Commit chronology authenticated against server-side anchors
- ☐ Clone triangulation run across available copies
- ☐ Anachronism review done on any disputed dating
- ☐ Similarity findings anchored to provenance and the fingerprint class
- ☐ Competitor comparisons run under expert protocol
- ☐ Findings stated to their corroboration rung

Red flags

- Repositories pushed in one batch long after their commit dates: Example 1.
- Dependencies or toolchains post-dating the commits that use them.
- Tokens created near resignation; access events after employment: Example 2.
- Histories beginning with a single squashed commit at a convenient date: Example 3's defendant.
- Force-push and history-rewrite events mid-dispute.
- Shared typos, dead code and comment fingerprints across "independent" codebases.
- Productions of file trees resisting native clone requests.

When to involve a digital forensic expert

At preservation, where platform log retention and reflog expiry set the clock; at collection, where full-history cloning, platform-layer export and hashing are craft; at authentication, where server anchors, triangulation and anachronism analysis separate history from assertion (Examples 1-3's machinery); at comparison, where similarity analysis and the confidentiality protocol need expert hands; and at reporting under CPR Part 35 with every finding on its corroboration rung. Through **e-discovery.uk**, code productions and their platform records sit in review beside the correspondence per guide 70.

§ 13 · COMMON QUESTIONS

Frequently asked questions

Can git really prove when code was written?

Git asserts it; corroboration proves it: commit dates are local and settable, so reliable dating stacks the server-side anchors: push times, PR and CI records: with clone triangulation and internal-consistency checks. Where those align, the dating is about as strong as digital chronology gets; where a repository stands alone, Example 1 is the caution.

A developer says someone else committed under their name. Possible?

Trivially: the author field is typed, not authenticated: so attribution climbs the ladder: which platform account pushed, from which IPs and devices, whether commits were signed, and whose working pattern the code matches. Signed commits and protected branches close the gap prospectively; forensically, the push record usually answers what the author field cannot.

The opponent deleted the repository. Is the history gone?

Rarely: git distributes complete history to every clone: developer machines, forks, CI checkouts, backups: and platforms retain deleted repositories for recovery windows while audit logs record the deletion itself. §8's map is the recovery plan, and the deletion event, dated against the dispute, is frequently worth more than the repository was.

How much code similarity is enough to prove copying?

No percentage answers it: volume of resemblance has innocent explanations (frameworks, generators, AI assistance); the probative class is the fingerprints: shared typos, dead code, arbitrary quirks: joined to provenance showing your version existed first. Example 3's single misspelled comment outweighed thousands of merely similar lines, which is the right intuition to keep.

Our code is commercially sensitive. Must we hand it to the opponent to fight a copying case?

No: the standard resolution is the §7 protocol: designated experts examine both codebases in controlled environments and report findings, with disclosure of the code itself confined or eliminated: confidentiality preserved on both sides while the comparison proceeds. Courts order and endorse such regimes routinely; refusing all examination while alleging copying is the position that does not survive.

Does AI-generated code change any of this?

It complicates similarity and authorship, not chronology: generated code produces innocent resemblance across unrelated projects (raising the bar to the fingerprint class) and dilutes what "the developer wrote it" means (assistant involvement is now a standard examination question): while the dating and integrity disciplines: hash chains, server anchors, triangulation: work unchanged. Expect provenance-of-authorship questions to grow, and keep the corroboration habits that answer them.

§ 14 · REFERENCE

Glossary

Clone
A full copy of a repository including its entire history.

Commit
A recorded change-set with author, dates, message and parents.

Fork
A platform-side copy under another account, linked to its origin.

Force-push
Overwriting server history with a rewritten local version.

Hash (SHA)
The commit's content-derived identifier; the integrity spine.

Personal access token
A credential for API/git access; its logs date use.

Pull request
A server-side proposed change with reviews and timestamps.

Reflog
A local log of reference movements; remembers rewritten states, expires.

Rebase / rewrite
Restructuring history; changes hashes from the edit onward.

Tag / release
A named marker binding a version to an exact commit.

Sources and authoritative references

The source-code disciplines in this guide draw on materials published by our laboratory and e-discovery practice, referenced here as sources:

- cflab.uk, digital forensics services (repository forensics, commit authentication, similarity analysis, CPR Part 35 reporting): cflab.uk/digital-forensics-services · guides library: cflab.uk/guides
- e-discovery.uk, EDRM Phase 02 · Preservation, Phase 03 · Collection and Phase 05 · Review (platform-log preservation; native repository collection; code in review): e-discovery.uk/edrm/preservation · e-discovery.uk/edrm/collection · e-discovery.uk/edrm/review
- e-discovery.uk, practice method and Knowledge Centre: e-discovery.uk · e-discovery.uk/knowledge
- Platform documentation: GitHub audit and security logs: docs.github.com · GitLab audit events: docs.gitlab.com · Git internals: git-scm.com/book
- PD 57AD, CPR Part 31 and CPR Part 35: justice.gov.uk, PD 57AD · justice.gov.uk, Part 35
- ICO, UK GDPR guidance: ico.org.uk · ISO/IEC 27037: iso.org, 27037

DISCLAIMER

This publication provides general information about source-code evidence as at August 2026. Platform capabilities, log coverage and retention vary by product, plan and configuration and change on vendors' schedules; verify the current position for the estate in the matter at hand. The worked examples are fictional. This guide is not legal advice, does not create a professional relationship, and should not be relied upon in place of advice on the facts of a specific matter from a qualified lawyer or a suitably qualified forensic practitioner. Links were live at the date of publication.

§ HOW A SPECIALIST LABORATORY CAN ASSIST

Working with Computer Forensics Lab

Source-code instructions at the laboratory run this guide's order: platform logs and tokens dealt with in the first call, because their half-lives set the clock; repositories collected as full clones with the server layer beside them; chronologies authenticated against push, PR and CI anchors with triangulation across every reachable copy; exfiltration chains built from clone and token logs into the device layer per guide 41; and similarity examinations anchored to provenance and run under confidentiality protocols where competitors face each other. Findings report under CPR Part 35 to their corroboration rung, and through **e-discovery.uk** the code record sits in review beside the correspondence that discussed it. Conflicts checks, confidential scoping discussions and fixed-fee estimates are routine; and if a developer resigned this week, the token revocation and log export are this afternoon's instructions.

INSTRUCT THE LAB

Speak to a forensic examiner, not a salesperson.

Describe your matter in confidence and an examiner will respond the same working day. Conflicts checks and scoping calls with US counsel are routine.

NEW ENQUIRIES
+44 (0)20 7164 6971

EMAIL
info@cflab.uk

E-DISCOVERY
e-discovery.uk

Computer Forensics Lab Ltd · Euro House, 133 Ballards Lane, Finchley, London N3 1LJ, United Kingdom
cflab.uk · Customer service +44 (0)20 7164 6915 · ICO ZB395023 · NPCC · ISO 17025 aligned practice · UK Gov Digital Marketplace