Guidedog Discussions GDS 0008

Publish documentation to GitHub Pages

STATE
committed
AUTHORS
Vikrant Rathore
UPDATED
2026-10-31
TYPE
Standards Track
CREATED
2026-10-12
DISCUSSION
Local review; no external discussion URL assigned
LABELS
cli, build, security, documentation

Record status

The catalog defines the review process and the meaning of each state. This numbered record forms part of the catalog.

Contents

1. Abstract 1
2. Motivation and scope 2
3. Review of the first draft 2
4. Specification 3
5. Rationale and alternatives 7
6. Security and operational considerations 7
7. Backwards compatibility 7
8. Acceptance criteria and implementation status 7
9. 0.6.1 reliability review 8
10. Open questions 9
11. Review history 9
12. References 9

NOTE

GDS 0012, Single executable release, updates the executable boundary for 1.0.0-beta.1. The release links this publisher into `guidedog` and exposes `guidedog publish pages` and `guidedog pages` maintenance commands. References below to a separate program describe the earlier design.

1. Abstract

`guidedog-pages` publishes a Guidedog site to GitHub Pages from a local machine. It builds the project with the installed `guidedog`, freezes the result, and sends the whole site to a branch of its own in one Git push. GitHub applies the push only if the branch still holds the commit the publisher read. The publisher then waits for GitHub Pages to deploy that commit, and calls the site live only when GitHub says so. Tokens stay in the system's credential store. A machine-owned binding decides which repository and branch a profile may publish to. `guidedog-pages watch` publishes again when the sources change. The same program is the `pages` publisher of GDS 0009, which `guidedog publish` runs through the process protocol.

GitHub Actions remains the second, independent route: the manual's workflow builds with a pinned Guidedog release and deploys with the job's short-lived token. Guidedog 0.6.0 ships both. The review of the first draft (below) replaced its transport, its build boundary, and its snapshot mechanism; the rest of its safety design stands.

2. Motivation and scope

A static site is useful only when readers can reach it. A project should be able to choose a repository, authorize one machine, and publish the same files it previewed, without keeping a maintainer's personal token in GitHub Actions. Teams that publish after every push keep the Actions route.

This record owns publication, credentials, remote status, and automatic updates. It reuses the project engine, templates, translations, diagnostics, and memory policy. It adds no Python and does not change document semantics. It does not provide a hosting service, a plugin marketplace, arbitrary build hooks, or a credential broker on Guidedog's servers.

GitHub.com is the only supported host. GitHub Enterprise Server needs separate qualification and host binding; a different API URL is not proof of support. DNS, custom domains, and repository creation remain outside the publisher. Only test code (`Roots::at` in `gd-pages`), never a configuration or CLI flag, can point a client at a local test server; a release command never sends its token anywhere but GitHub.com.

3. Review of the first draft

The first draft (2026-10-12) was reviewed before implementation. Its route split, its refusal to force a branch, its journal, its credential rules, and its outcome states were sound and are kept. Four parts would not have worked as written:

Table 15 Findings and decisions

Draft	Finding	Decision
Upload with the Git Database API: one request per blob and tree, then a non-forced ref update.	GitHub limits content-creating requests to 80 a minute and 500 an hour (secondary rate limits). A first publication of a few hundred pages and their assets would take hours; base64 JSON adds a third to every byte. The draft itself admits the ref update is not a compare-and-swap.	Push with Git's smart HTTP protocol: one `git-receive-pack` request carries a pack of every object the branch lacks and the update from the observed commit to the new one. receive-pack applies an update only while the reference still has its old ID: a true compare-and-swap. `gd-git` writes the objects and messages; no git program runs.
A "thin embedded renderer" links the project engine into `guidedog-pages`.	A second full engine (Typst, Graphviz, tree-sitter) to build, qualify, and ship.	`guidedog-pages` builds with the installed `guidedog`; linked into `guidedog`, the publisher uses the host's own build. `guidedog publish` builds with its own engine and gives the frozen site to the publisher. One engine; the publisher links none.
Freeze the output "under the build coordinator's output lock".	The project coordinator has no output lock: a build publishes by switching folders, and adding a cross-process lock to every build is out of proportion.	Build into a folder only this publisher writes (its target's cache folder), under the target's lock, and freeze it there; a site built elsewhere (`site =`) is copied. `publish` re-reads every file before it writes, and packing checks each file's Git ID again.
Device login "out of the box".	The device flow needs a registered GitHub App or OAuth App's client ID, a maintainer decision the draft left open.	The client ID is a compile-time environment variable (`GUIDEDOG_PAGES_CLIENT_ID` via `option_env!`) or `--client-id`. Releases built without one use `auth import` to

Draft	Finding	Decision
		store a fine-grained token, and `auth login --client-id` works with an app the user registered.

Two smaller corrections: status is correlated through the Pages builds API, which lists each build's commit (a deployment of a branch has no Actions OIDC deployment to read); and publication commits carry a trailer, `Guidedog-Publication: GENERATION`, by which the publisher tells its commits from others on the branch.

4. Specification

4.1. Two publication routes

GitHub's Pages deployment endpoint requires an Actions-issued `oidc_token` and an artifact of the repository. A local token supplies neither. The local route therefore publishes a branch, and Pages deploys the branch.

Table 16 Publication routes

Route	Local publisher	GitHub Actions
Build	The installed `guidedog` (or `guidedog publish`'s own engine)	A pinned Guidedog release
Transfer	One smart-HTTP push of a pack and the branch update	Official Pages artifact action
Authorization	A stored token (fine-grained, or device flow) and a machine binding	Job token and Actions OIDC identity
Pages source	Dedicated branch, root folder	GitHub Actions
Automatic update	`guidedog-pages watch`, or a machine scheduler	Push to the chosen source branch

A repository has one Pages publishing source. `publish` reads it and refuses a site deployed by a workflow or from another branch or folder; `setup` changes it, after asking. The publisher never silently replaces an Actions configuration.

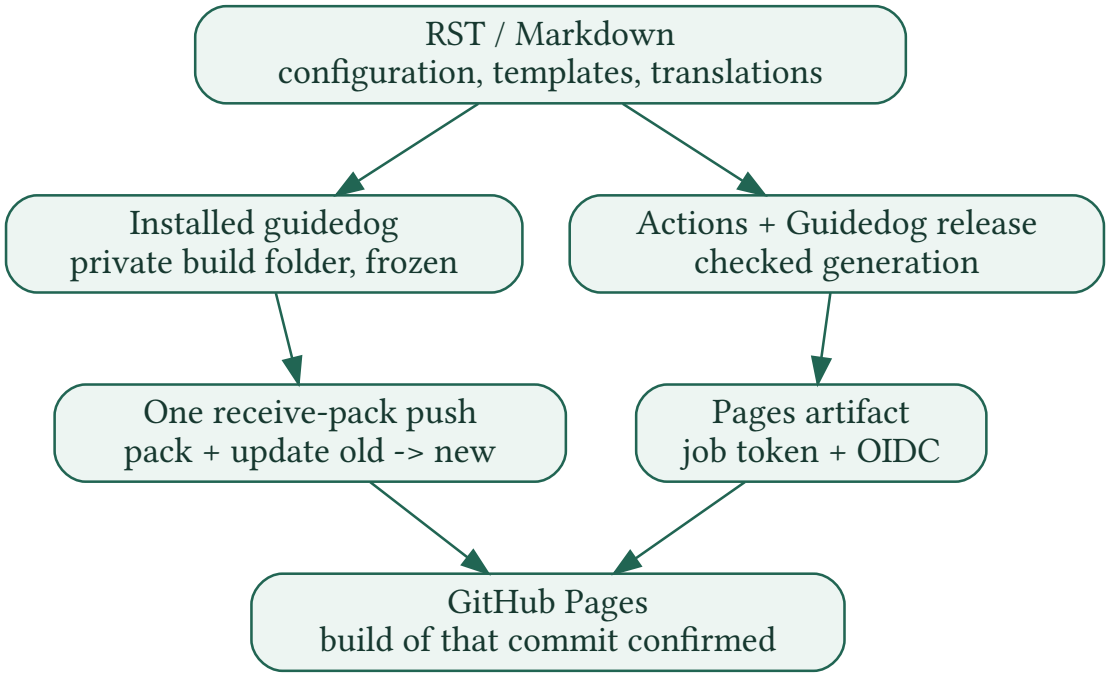

Fig. 9 Both routes publish a successful Guidedog generation.

4.2. Programs and libraries

Table 17 Where the publisher lives

Part	What it does
`gd-git`	Git object IDs, trees from a list of files, commits, packs (zlib), and the receive-pack advertisement, request, and report. No networking, no git program. `PackWriter` removes an unfinished pack when it is dropped.
`gd-fetch`	`send`: any HTTP method, headers, a body in memory or streamed from a file, bounded answers, cancellation, and no redirects (an `Authorization` header reaches only the URL it was meant for). HTTP and TLS run in-process (`ureq` and `rustls`), so no `curl` program, OpenSSL or system certificate store is needed.
`gd-pages`	The `pages` publisher of `gd-plugins`: configuration, credential stores, bindings, journals, locks, the GitHub client, the device flow, and the publication itself. It also adds `guidedog publish pages` (guided) and `guidedog pages workflow` to `guidedog`.
`guidedog-pages`	The command (`cmd/guidedog-pages` over `gd_pages::command`); started with `--guidedog-protocol`, it serves GDS 0009's process protocol instead.
`guidedog publish`	Builds with the engine, freezes, and drives the `pages` publisher, linked in or registered.

`gd-pages` depends on `gd-fetch`, `gd-git`, `gd-plugins`, and `gd-toml`, and nothing of the document engine. A publisher is either linked into the `guidedog` binary or registered explicitly: `guidedog plugin register pages PATH`. Neither a project nor a working directory selects code, and a normal build never starts a publisher.
Credentials stay in the system's store. On Linux and other Unix desktops the Secret Service is reached through the `secret-tool` program with the token on standard input and output, never on a command line. macOS uses the Keychain and Windows the Credential Manager through small FFI modules. There is never a plaintext fallback, and `Secret` zeroes itself on drop. Settings are plain data (`Settings`, `Roots`), so only code, never the environment, can point a release away from GitHub.

4.3. Configuration and binding

`pages.toml` records non-secret settings; `guidedog-pages init` writes it. Unknown settings and invalid values are errors with their line.

```
schema = 1
repository = "example/project"
branch = "gh-pages"
project = "docs"          # or: site = "build/site"
strict = true
profile = "default"
site_url = "https://example.github.io/project/"
wait_seconds = 300

[watch]
debounce_ms = 1000
max_wait_ms = 10000
```

Paths are relative to the file. `site_url` defaults to where GitHub serves the repository, and is the build's `html_baseurl`. `profile` only names a stored token; the profile `env` is never accepted here (see credentials).
A *binding*, kept in the machine's state folder (`plugins.config_dir()/pages`, readable only by the user), authorizes a profile for one repository, by its GitHub ID, and one branch, for one account ID. `bind` creates it after asking. Before it asks, it shows the account, the repository and its ID, its visibility, the branch's state, and the Pages settings. Publication refuses the following, before it moves anything: a target with no binding; a profile other than the bound one; a token of another account; a repository whose ID is not the bound one (deleted and made again, or another took its name); a renamed or transferred repository, including an API redirect; and the repository's default branch. A project edit can name a target, but never authorize one.
A branch that holds commits without the publication trailer is someone else's work. `bind` refuses it unless `--adopt` is given, after which the binding records the adopted commit and the next publication replaces the branch's files.

4.4. Commands

```
guidedog-pages init --project=docs --repository=example/project
guidedog-pages auth import --profile=default
guidedog-pages auth login --client-id=APP_ID --scope=public
guidedog-pages auth status
guidedog-pages bind [--adopt] [--yes]
guidedog-pages setup [--yes]
guidedog-pages doctor
guidedog-pages publish [--dry-run] [--wait=SECONDS] [--verbose]
guidedog-pages watch
guidedog-pages status [--operation=COMMIT]
guidedog-pages recover
guidedog-pages unbind
guidedog-pages auth logout
```

`init` writes configuration and changes nothing at GitHub. `setup` points Pages at the branch's root, creating the Pages site when there is none. It needs the Pages and Administration permissions; manual setup in the repository's settings works too. GitHub configures Pages only for a branch that exists. So a target without a Pages site is still published to, and the publication ends `Accepted` with `pages.no-site`. Run `setup` then. (GitHub turns Pages on by itself when a `gh-pages` branch first arrives; the qualification below saw it do so.) `setup` refuses a branch that does not exist yet, and says to publish first. `doctor` checks the configuration, the builder, the token, the account, the repository and push permission, the binding, and the Pages source, and writes nothing. `--dry-run` builds and reads the target, and writes nothing: no object, no ref, no setting, no credential.

4.5. Credentials

Tokens are stored under the service `guidedog-pages` with the profile as account: in the macOS Keychain (Security framework items), the Windows Credential Manager (generic credentials), or the Secret Service of a Linux desktop (reached through `secret-tool` over standard input and output, so the program links no desktop C library and runs where `secret-tool` is absent). `profiles.toml` records each profile's account login, ID, method, and date, never the token. A missing or locked store is a problem naming the store; there is no plaintext fallback.

`auth import` reads a token from a hidden terminal prompt or from standard input, never from an argument. `auth login` runs the device flow. It shows GitHub's verification address (only an address on github.com is shown) and code, opens a browser only with `--open`, polls at the interval GitHub asks for, slows down on `slow_down`, and stops on expiry, denial, or Ctrl+C. Both check the token with GitHub and refuse to give a profile to another account without `--replace`. `--scope=public` asks an OAuth App for `public_repo` instead of `repo`.

`GUIDEDOG_PAGES_TOKEN` gives a token to one process, never stored. It is used only when chosen explicitly: `--credential=env`, or `GUIDEDOG_PAGES_CREDENTIAL=env` for runs the command line cannot reach, such as `guidedog publish`. An unattended run never starts a device login. `auth logout` deletes the local token and says how to revoke it at GitHub; local deletion is not revocation. Token bytes the publisher owns are cleared when it is done with them; copies a transport library makes are outside that promise.

Table 18 Permissions by operation

Operation	Fine-grained token / GitHub App	OAuth App token
Read the account, repository, and branch; push the branch	Metadata: read; Contents: read and write	`public_repo` (public repositories) or `repo`
Read the Pages settings and builds (confirmation)	Pages: read	`repo`; without it, publication ends as `Accepted`
`setup`: create or change the Pages source	Pages: write and Administration: write	`repo` and the repository's admin role

Organization policies (SSO, app approval, rulesets) are reported as GitHub states them: a refused permission names the permission to grant, and a ruleset refusal of the push names the ruleset as the cause.

4.6. Publication

A publication runs under the target's lock, held by one process at a time on the machine (`KEY.lock`, a kernel lock); another machine is held off by the compare-and- swap.

1. Build into the target's private folder (or copy `site`), and freeze it: every regular file, path, size, and SHA-256 (GDS 0009's snapshot). Links, special files, paths Git or a static host cannot serve, case collisions, a missing `index.html`, and sites over 1 GiB, 100,000 files, or a file of 100 MiB are refused before anything is sent.
2. Refuse while a journal of the target is unsettled. Read the Pages source; read the branch's commit `H` from the receive-pack advertisement; read `H`'s tree and message, and refuse a commit without the trailer that was not adopted; read `H`'s files, recursively, with their blob IDs.
3. Take each file's Git blob ID, add an empty `.nojekyll` (so `_static` is served), and build the trees. When the root tree equals `H`'s, the target is `Unchanged`: nothing is written.
4. Make commit `C`: the new tree, parent `H`, by the account at its private `ID+login@users.noreply.github.com` address, at `SOURCE_DATE_EPOCH` or now, with the trailer `Guidedog-Publication: GENERATION`.
5. Write a pack of the blobs the branch lacks (re-reading and checking each file), every tree, and `C`, then the request `H -> C` for `refs/heads/BRANCH`, to the target's cache folder.
6. Write the journal (repository and ID, branch, `H`, `C`, generation; no secret).
7. Send the request. A report of `ok` is acceptance. `ng` is refusal, and the journal is dropped: a refusal changes nothing. If the branch no longer has `H`, the problem is a conflict: another publication came first. A lost answer is settled by reading the branch again: `C` is acceptance, `H` means nothing was published, anything else is a conflict. When the branch cannot be read, the outcome is `Unknown` and the journal stays for `recover`.
8. On acceptance, record `C` as the target's last publication and drop the journal. Poll the Pages builds for `C` until `wait_seconds`: `built` is `Live`, `errored` is `Failed` (the branch has the site, the previous deployment stays), a queued or running build is `Pending`, and none, or builds the token may not read, leave it `Accepted`.

The update of the branch is the publication's boundary. Objects GitHub received for a refused update are unreferenced; readers of the branch see the old tree or the new one. This does not claim that every CDN edge changes at once.

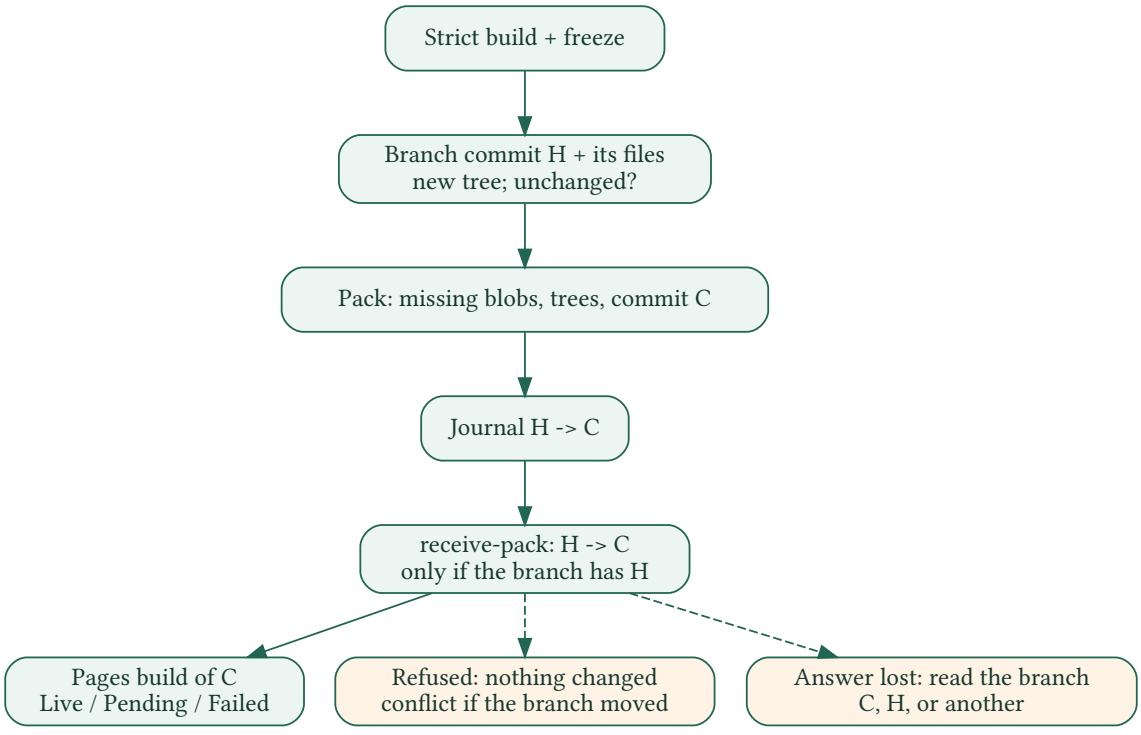

Fig. 10 The branch moves only in the push, and only from the commit read.

4.7. Outcomes

`publish`, `status`, and `recover` end in one of GDS 0009's states, and exit with: 0 for `Live` and `Unchanged` (and a dry run); 1 for `Failed` (nothing published, or the deployment failed); 2 for a usage or configuration problem; 3 when memory ran out; 4 for files or the network; 6 for `Accepted` and `Pending` (the branch moved, deployment not confirmed: run `status` later); and 7 for `Unknown` (run `recover`). Problems name their code (`pages.conflict`, `pages.ruleset`, `pages.credential`, `pages.unsettled`, …) and the next step. JSON diagnostics carry no response body, header, or credential.

4.8. Watching and recovery

`watch` is in the foreground and opt-in. It publishes once, then stamps the source folder (paths, sizes, and times of its files, without build output, caches, and hidden folders such as `.git`) every half second, and publishes once the sources have been quiet for `debounce_ms`, or `max_wait_ms` after the first change. Each round frees what it made, so a long watch does not grow. A failed build or publication is reported, and the site stays as it was. A refused or missing token pauses publication without opening a browser. Network failures are retried after 2, 4, 8, 16, and 32 seconds. Ctrl+C stops it between steps; a push already sent is settled as above. It watches this machine's files. Publication after a push to GitHub belongs to the Actions route. The manual gives launchd, systemd, and Task Scheduler examples.

`recover` reads the journal and the branch, and settles as a lost answer is settled. It never moves the branch back. Without a journal it reports the last publication's deployment. A damaged journal is a problem with directions, not a reason to touch the branch. A future rollback would publish an older tree as a new commit.

5. Rationale and alternatives

The smart-HTTP push is what `git push` does, so its limits are Git's: one request, compressed objects, and only those the branch lacks. Its update is a true compare-and-swap, which the Git Database API's fast-forward check is not. It needs no git program, no shell, and no credential helper; the token goes in an `Authorization` header of two requests to github.com. A future transport must keep the same transaction and credential contracts.

Building with the installed `guidedog` keeps one engine and one qualification; `guidedog publish` offers the in-process build to those who prefer it. A private build folder freezes a generation without a lock in the build coordinator; a copy covers sites built elsewhere.

For teams, the official Actions route offers environment protections, short-lived authorization, and updates on push. A personal token in repository secrets would broaden access unnecessarily.

6. Security and operational considerations

Project sources, templates, paths, API answers, and remote file names are untrusted input. Answers are bounded before they are parsed (4 MiB, 64 MiB for a tree listing, 16 MiB for Git messages), JSON nesting is limited, duplicate keys are refused, and text from GitHub is escaped before it reaches a terminal. TLS verification is never turned off. Requests that carry a token never follow a redirect. Only GitHub.com is reached, except by a test build. The device flow's address is shown only when it is GitHub's.

State files are written beside their place, synced, and renamed over it, with owner-only permissions. Tokens are never written to a file, an argument, a journal, a log, or a crash report. The publisher runs no Python, repository hook, shell command, or Git credential helper.

Publishing a private repository's site does not make the site private; `bind` says so, with the repository's visibility. The publisher publishes the frozen build output only, never the repository root.

7. Backwards compatibility

Guidedog's existing commands, `conf.toml`, rendering, catalogs, and templates are unchanged. Without `guidedog-pages` or `guidedog publish`, no GitHub request or authentication happens. The Actions workflow keeps working. `pages.toml`, the state files, and the journal have their own schema versions.

8. Acceptance criteria and implementation status

Guidedog 0.6.0 ships `guidedog-pages` in the `guidedog` archives and the Homebrew formula, beside `guidedog`.

Table 19 Criteria and evidence

Criterion	Status
Explicit ownership, checked bounds, bounded HTTP, cancellation, no Python.	Done. Ownership and bounds are Rust's; HTTP answers are bounded by `Options::max_bytes` and the timeout; `builds_run_without_python`.
A fresh site, a changed site, byte-identical output that writes nothing, removed files, underscore paths.	Done, against a real `git receive-pack` behind a local GitHub stand-in (`crates/gd-pages/tests/publish.rs`); `git fsck --strict` checks every push.
Concurrent writers, a branch of other work, lost answers, journals, failed and pending deployments.	Done in the same tests: conflict, adoption, lost-before and lost-after, recover, `errored`, `building`, and no build.
Unbound, renamed, re-created, and default-branch targets; another account.	Done.
The same outcomes linked in and through the process protocol.	Done: the `guidedog-pages` program, driven through `gd-plugins`, publishes as the linked publisher does.
Device flow: pending, slow down, success; denial, expiry, cancellation.	Done against the stand-in. The release ships no app: an app registered by the maintainers, with its permissions published, is still to be decided.
Real Keychain, Credential Manager, and Secret Service.	The Keychain is tested on macOS; Windows qualification requires a real Credential Manager round trip. Linux qualification starts an isolated D-Bus session and GNOME keyring, and requires Secret Service storage and removal. Headless environments still refuse unavailable stores without a plaintext fallback.
A real repository on GitHub.com.	Done on 2026-10-13, with a disposable public repository, `insanai/guidedog-pages-check`, made for the purpose, and the maintainer's OAuth token (`repo` scope): `init`, `bind`, a first publication that made `gh-pages` (3.3 s; `Accepted`, `pages.no-site` until GitHub enabled Pages), `status` `Live` once the Pages builds API listed that commit as `built`; a changed page `Live` after 28 s of waiting; the same site again `Unchanged`; a publication through `guidedog publish` and the process protocol; one with a token stored in the macOS Keychain; `auth logout`. The served site showed each change and `_static/`; a clone passed `git fsck --strict`, with one linear commit per publication, each with its trailer. A conflict on GitHub itself was not staged; receive-pack's refusal is qualified against git.
The manual in every maintained language; scheduler examples; release on all five platforms.	The English manual is complete; translations ship with 0.6.0. Release qualification covers the five platforms with an offline smoke test.

The maintainer accepted the implemented scope on 14 October 2026. The credential adapters are qualified against the native stores. A maintainer-owned device-flow app is a separate deployment decision; importing a token or using an explicit client ID remains supported. See Plugin design acceptance, 14 October 2026.

9. 0.6.1 reliability review

Credential copies remember their allocation owner and are wiped before release. Native-store allocations are checked before crossing C bindings. Local metadata updates hold an exclusive lock across the whole read/replace operation, including separate sessions in one process. Dry runs have separate build folders.

An accepted push remains accepted if saving the local report fails. The journal is deleted only after that report is durable, so `recover` can reconcile a lost reply or local disk error. It never resets the remote branch or repeats a push blindly. The plugin host now bounds whole requests; a timeout after a possible remote write reports `Unknown` and asks for recovery. These are executable regression fixtures.

The manual teaches the native publisher interface using this implementation and the smaller folder publisher. All seven editions share the tested source listings.

10. Open questions

Who registers and owns the device-flow app, and whether it is a GitHub App (repository selection, fine-grained permissions) or an OAuth App (simpler, broader scopes). The Pages builds API reported the branch deployment in the qualification. Whether it does on every plan, or the deployments API is needed as a second source, is still to be seen. Enterprise Server, custom domains, and forward rollback each need their own qualification.

11. Review history

2026-10-12: created at the maintainer's request. The design separates local branch publication from Actions artifacts because the Pages deployment API requires Actions OIDC. The manual gains the currently usable Actions route.
2026-10-13: reviewed and implemented for Guidedog 0.6.0. The review replaced the Git Database API with one smart-HTTP push (secondary rate limits; a true compare-and-swap), built with the installed `guidedog` instead of a linked engine, froze a private build folder instead of an output lock that does not exist, and made the device-flow app a build setting. The record now describes the implementation.
2026-10-15: the guided `guidedog publish pages` and `guidedog pages workflow` join the commands, linked into `guidedog`.

12. References

- Guidedog: CLI and documentation toolchain, the command line and toolchain.
- Memory budgets and what a build may read, memory and trust boundaries.
- Sphinx-style projects and recoverable publication, local project publication.
- Discussion catalogs for any project, the catalog used to manage this record.
- Native plugin API and registry, the plugin foundation.
- GitHub Pages publishing sources.
- Pages REST API.
- Rate limits for the REST API, secondary limits.
- Git's HTTP protocol and pack format.
- OAuth device authorization.
- GitHub App user tokens.
- GitHub Pages limits.
- `docs/manual/howto/github-pages.rst`, both routes, step by step.
- `crates/gd-pages`, `crates/gd-git`, and `crates/gd-fetch` (linked into `cmd/guidedog` per GDS 0012).

Lifecycle event

```
2026-10-12: prediscussion -> discussion. Open the maintainer-requested
publishing plugin design for review
```

Lifecycle event

```
2026-10-14: discussion -> accepted. Maintainer requested acceptance after
review of the implemented 0.6.2 scope.
```

Lifecycle event

```
2026-10-14: accepted -> committed. Implemented scope reviewed against source
and published qualification.
docs/gds/evidence/plugin-acceptance-20261014.rst; successful 0.6.2 release
run 38434657070.
```