

Ship & operate

Branching & Delivery Flow

🕒 3 min read · Updated 27 September 2026 · Verified against d2b3999 on 2026-09-26

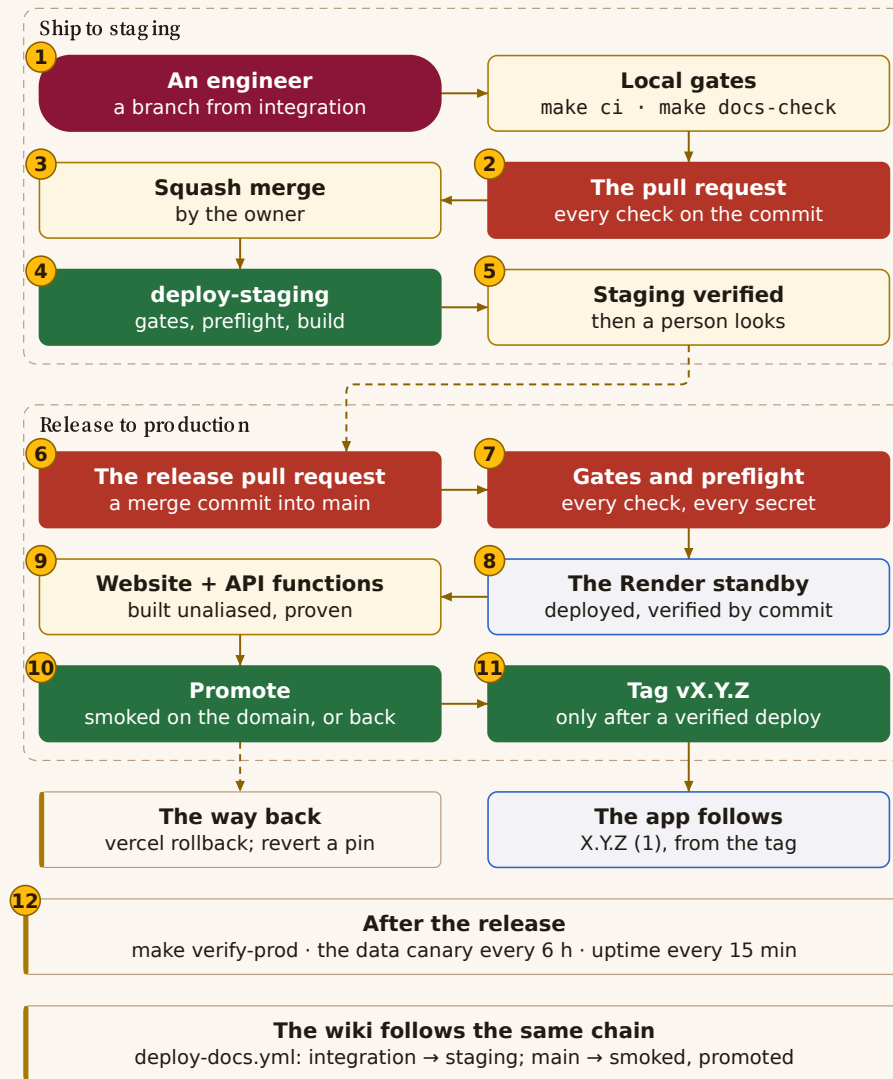
The branch model: feature branches into integration, release pull requests into main, required checks and pull-request hygiene.

<https://docs.gurbanisoul.com/process/branching/>

The whole journey, from a branch to production and the App Store, as the pipeline runs it — press *Next* on the site to walk it:

11 · The delivery pipeline

From a branch to staging, to a release on main, to production and the App Store — every step gated by CI, verified by commit, and reversible



■ a person □ component in this project ■ gate: fails the build or the deploy
 ■ proven, verified state □ external system □ explanation
 — flow: data or control moves this way
 --- reference: pins, reads or derives from

Source of truth: .github/workflows/deploy-production.yml ·
 .github/workflows/deploy-staging.yml · docs/process/branching.md ·
 docs/process/release.md

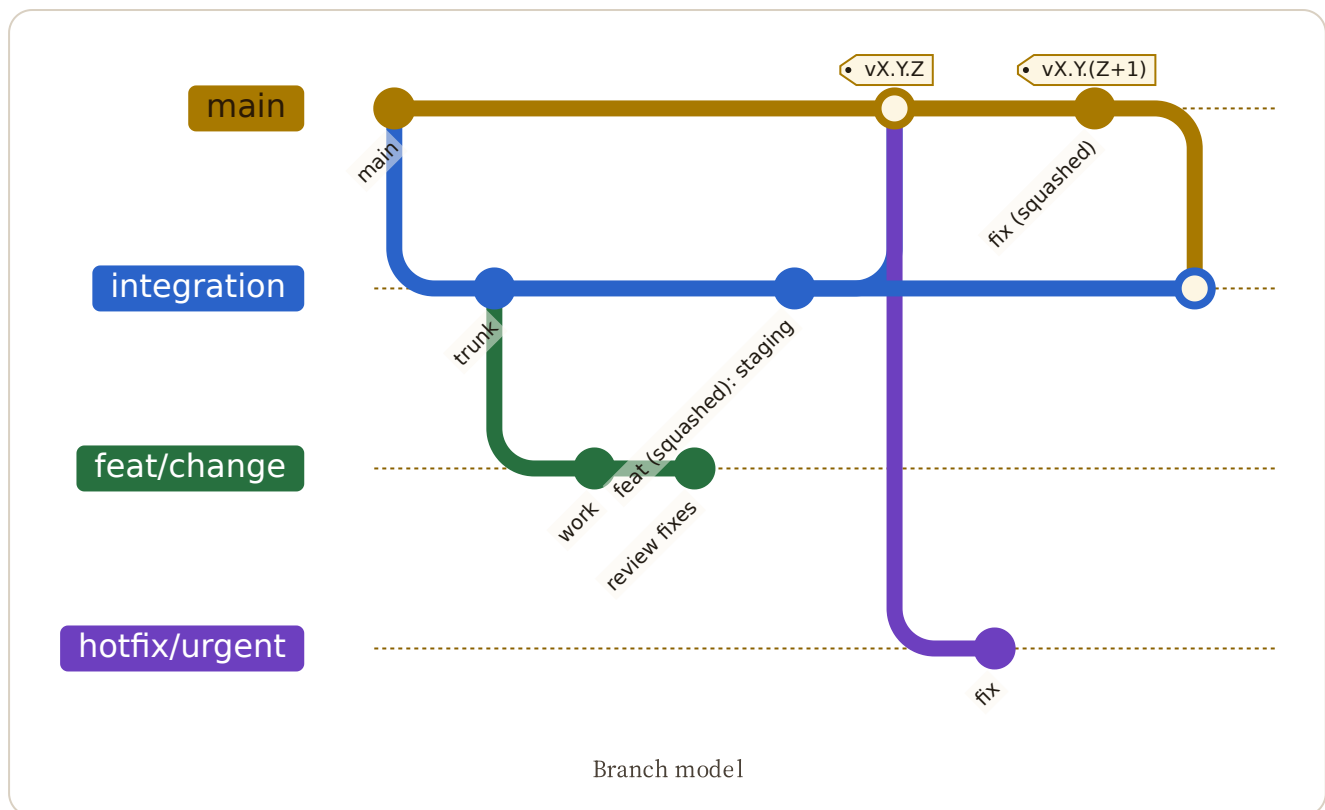
v1.3.10 · verified 2026-09-27 · d85c2ad

Sri Guru Granth Sahib Ji — Knowledge Base · engineering wiki

Poster 11 — The delivery pipeline: ship into integration and onto staging, release into main and onto production, tag, the app follows, then the canary and uptime watch

- 1. Branch and run the local gates.** Branch feat/... or fix/... from integration, the trunk. Before opening a pull request run make ci (versions, dataset check, the server tests, the contract, the docs gates) so the pull request's checks are a confirmation, not a discovery.

2. **The pull request.** Every workflow on the CI gates map runs on the commit: web-ci, scripture-integrity, version-consistency, security, pr-hygiene, e2e and the docs job. A change to scripture handling, search, verification or the dataset pin completes the scripture-safety checklist in the template.
3. **Squash merge.** Feature and fix pull requests are squash-merged into integration so the trunk carries one clean commit per change. Only the owner merges (the ruleset requires CODEOWNERS); the engineer hands over the exact command.
4. **deploy-staging.** The push runs deploy-staging: the gates job waits for the core checks and playwright on the commit, preflight checks the staging secrets, the API functions are generated from the pinned database (each context's slice proven by the slicer), vercel build proves what each function bundles, the deployment is aliased to sggs-staging.vercel.app and the CDN cache is purged.
5. **Verify staging, then a person looks.** verify_functions.py checks every function's /readyz reports this commit and exactly its contexts; the gateway probes prove each context answers through the real rewrites with its own X-Service; the web smoke runs; the whole golden contract passes through the gateway. Then the owner reviews the staging site, logged in to Vercel, before any release.
6. **The release pull request.** make release VERSION=X.Y.Z bumps the unified version everywhere and stubs the CHANGELOG section; a dataset change arrived earlier as a reviewed lock bump with a regenerated contract. The pull request goes from integration to main and is merged as a merge commit — never squashed — so main stays a descendant of integration.
7. **gates, preflight, approve.** deploy-production waits until every check the main ruleset requires, plus playwright, has succeeded on the exact merge commit; preflight proves every production secret exists; approve is an optional human click when the production-approval environment has reviewers. Nothing has been deployed yet.
8. **The Render standby, verified by commit.** deploy-api triggers the Render deploy hook with ref=SHA and waits until /api/health.commit equals the commit and the health checks are all true; verify-api then samples compositions and proves exact lines, present headings and 404 on the permanent comp_id gaps. Since 1.3.10 production's /api is answered by the API functions in the web deployment; the Render API is kept deployed and proven as the standby. The web is untouched until this passes.
9. **The web and the API functions, built unaliased and proven.** vercel build and deploy with --skip-domain produce a deployment nobody can reach yet: the website and the API functions together. Its functions are checked at the commit, the golden contract is replayed against it and the @smoke tests run. A failure here leaves users on the previous deployment.
10. **Promote — with a way back.** vercel promote puts the deployment on gurbanisoul.com, the CDN cache is purged so no stale API body outlives its dataset, and the @smoke tests run against the public domain. If they fail, the job runs vercel rollback to the recorded previous deployment — the website and the API functions go back together — and opens an incident issue. Sending the API to the Render standby is a routing change in gateway/routes.json; a dataset rolls back by reverting the lock bump.
11. **Tag, and the app follows.** Only after a verified deploy does the release job create the tag vX.Y.Z and a GitHub Release from the CHANGELOG section. The app repository then vendors the contract at that tag, sets MARKETING_VERSION and uploads X.Y.Z (1) with CHANNEL=appstore; scripts/release/check_release_complete.py proves web, API and the binary report the same number from the same commit.
12. **After the release.** make verify-prod proves what production serves by commit identity. Every six hours the data canary proves the served scripture equals the pin at the origin and through the CDN and replays the golden contract; every fifteen minutes uptime probes /api/health and the App Store URLs. The wiki deploys through the same chain in deploy-docs.yml.



Rules

- **integration is the trunk.** Branch feature/* or fix/* from it; open a PR back into integration. Merging to integration deploys **staging** (the web with the API as Vercel functions inside it, ADR-0011). TestFlight uploads are **manual** and happen in the app repository (gurbani-soul-ios) — no merge here uploads a build.
- **main is production.** It only ever receives a **release PR from integration** (or hotfix/*). The version-consistency gate enforces the source branch. Merging to main tags vX.Y.Z, cuts a GitHub Release, and deploys production.
- **hotfix/*** branches from main for urgent production fixes, then is back-merged into integration.

Merge strategy (important)

PR	Merge method	Why
feature/*, fix/* → integration	Squash	one clean commit per change on the trunk
integration → main (release)	Merge commit	main stays a descendant of integration , so branches never diverge and no back-merge is needed
hotfix/* → main	Squash, then merge main → integration	keeps the trunk current
wiki PRs that move a verified stamp → integration	Merge commit	a stamp names a commit on the PR branch; a squash drops it from history and the stamp stops resolving

`main` therefore does **not** require linear history (a squash/rebase release would rewrite SHAs and make every later release PR conflict).

Required checks

Applied from `.github/rulesets/` via `scripts/gh/apply_rulesets.sh` (the JSON files are the source of truth — keep this list in step with them). The script **merges** each file into the live ruleset (`scripts/gh/merge_ruleset.py`): keys GitHub added that the file does not mention are kept, never reset to a weaker default. Run it with `--dry-run` first; it prints exactly what would change.

- `main`: PR + 1 approval + CODEOWNERS + resolved threads + strict status checks `python, frontend, integrity, versions, secrets, static-analysis, docs`; no deletion or force-push. `docs` gates the **merge** only: `deploy-production` waits on every other required check but not on the wiki's (`wait_for_checks.py --exclude docs`), which deploys the wiki on its own. **No linear-history rule** (see Merge strategy).
- `integration`: PR + CODEOWNERS + checks `python, frontend, integrity, secrets, docs` (the wiki job, ADR-0012); no deletion or force-push.
- Tags `v*`: no deletion or force-push.
- `playwright` and the iOS jobs are not required checks, but `deploy-staging / deploy-production` wait for `playwright` on the exact commit before deploying.

PR hygiene

Conventional-commit titles (`fix(data): ...`), small diffs, Closes `#NN`, the scripture-safety checklist in the PR template completed for any search, verify, display or dataset-pin change.