Common Version Control Procedure

Purpose: Provide a clear, consistent, and safe process for teams using Git to collaborate, review code, and ship reliable releases. Applies to all repositories unless an approved project-specific variant is documented.

1. Branching & Environment Strategy

Default: GitHub Flow (simple, continuous delivery)

- main: always deployable; protected (no force-push, status checks required).
- feature/*: short- lived branches for work items; created from main; merged via PR.
- hotfix/*: urgent fixes branched from main; PR with expedited review.
- release tags: semantic tags (e.g., v1.4.0).

Alternative (for long-running enterprise releases): GitFlow (appendix).

2. Daily Workflow (Author Checklist)

Before starting

1) git checkout main
2) git pull --ff-only origin main

Create a feature branch

3) git checkout -b feature/<short-slug> # e.g., feature/landing-fallback

Work locally

4) Make small commits; run tests and lints locally.

Synchronize

5) git fetch origin
6) git rebase origin/main # keep a linear history; resolve conflicts

Publish & open PR

7) git push -u origin feature/<short-slug>
8) Open a Pull Request to main; fill PR template.

Address review

9) Push more commits or use --fixup; rebase if requested.

Merge

10) Squash merge (preferred) or rebase-merge once checks pass.
11) git branch -d feature/<short-slug> (and delete on remote).

3. Commit Message Convention (Conventional Commits)

Format: <type>(<scope>): <summary>
Types: feat | fix | docs | style | refactor | perf | test | build | ci | chore | revert
Examples:
• feat(auth): add OAuth device flow
• fix(api): handle 404 on dataset lookup
• docs(readme): add mapping guide
Rules:
• Use imperative mood; keep summary ≤72 chars.
• Include body with motivation or breaking-change notes.
• Use BREAKING CHANGE: in body when applicable.
• Reference issues: “Fixes #123” or “Refs #456”.
Signing (recommended): enable GPG or SSH commit signing.

4. Pull Requests & Code Review

Opening a PR
• Small, focused changes (ideal review time ≤15 min).
• Fill PR template (what/why/how/test plan/roll-back).
• Link issues and screenshots or logs when relevant.

Required checks (examples)
• Lint, unit tests, type checks, build, vulnerability scan.
• Coverage threshold not decreasing beyond agreed %.

Review policy
• At least 1 reviewer (2 for risky areas).
• SLA: first response within one business day.
• Authors respond to every comment or resolve with changes.
• No self-approval unless repo owners explicitly allow.

Merging
• Prefer Squash & Merge for readable history.
• Avoid merge commits on feature branches (use rebase).

5. Branch Protections & Access

Protections on main
• No force push, require up-to-date branch, require status checks.
• Require code owners review for protected paths.
• Enforce signed commits (optional but recommended).
Access
• Use least privilege; rotate credentials on role changes.
• Require 2FA for all collaborators.

6. Releases, Versioning & Changelog

Versioning

- Semantic Versioning: MAJOR.MINOR.PATCH
- Tag format: vX.Y.Z; signed annotated tags preferred.

Release process

1) Ensure main is green.
2) Update CHANGELOG.md (Keep a Changelog format).
3) Bump version (automate with Release Please or semantic- release).
4) Tag: git tag -a vX.Y.Z -m "Summary"; git push origin vX.Y.Z
5) CI creates artifacts (container, package, build) and release notes.

Hotfixes

- Branch hotfix/x.y.z from main; PR with expedited review; tag patch release.

7. CI/CD Integration

- PR checks: lint, tests, SAST/Dependency scan, build.
- On main: build & deploy to staging automatically; manual approval to prod.
- Use environment protection rules (reviewers, secrets).
- Rollback plan documented per service (git revert or deploy previous tag).

8. Large Files, Secrets & Sensitive Data

Large files

- Use Git LFS for binaries over ~10 MB.
- Do not commit generated artifacts; add to .gitignore.

Secrets

- Never commit secrets (.env, keys, tokens).
- Use a secret manager (GitHub Actions secrets, Vault, etc.).
- If leaked, rotate immediately and purge history (BFG/TruffleHog).

9. Database & Migration Changes

- Migrations must be backward compatible for zero- downtime deploys.
- Review includes data volume, runtime, and rollback path.
- Tag release only after migration successfully applied in staging.

10. Handling Conflicts & Rebasing

Preferred: rebase feature branches on origin/main.

Commands:

- git fetch origin
- git rebase origin/main

If conflicts:

• Resolve in editor; run tests; continue: git rebase --continue
• If stuck: git rebase --abort and sync with teammate.
Merging from main into feature is acceptable for complex conflicts (document why).

11. Repository Structure & Conventions

• README.md: setup, run, test, deploy, contribution guide.
• CONTRIBUTING.md: how to branch, commit, open PR, review expectations.
• CODEOWNERS: critical paths mapped to reviewers.
• LICENSE, SECURITY.md, CODE_OF_CONDUCT.md.
• .github/ISSUE_TEMPLATE & PULL_REQUEST_TEMPLATE.
• docs/ for architecture, ADRs, and examples.
• Keep modules small; consider monorepo tooling if applicable (Nx, Turborepo).

12. Issue Tracking & Linking

• Every non- trivial change should map to an issue/ticket.
• Reference in commits and PRs (“Fixes #123”).
• Use labels (type, area, priority) and milestones for planning.

13. Quality Gates & Definition of Done

A change is “Done” when:
• All required checks pass.
• At least one review approval.
• Docs/tests updated.
• Feature flag or config added if rollout needed.
• Deployed to staging and verified (if applicable).

14. Rollbacks & Incident Response

• Fast rollback: deploy previous tag or git revert merge commit.
• Keep a runbook with commands and owner contacts.
• Post- incident: root cause, follow- ups, and PRs linked in ticket.

15. Security & Compliance

• Enable Dependabot/renovate for dependency updates.
• SCA/SAST in CI; block on critical severity.
• Enforce branch protection and least- privilege access.
• Keep an audit trail (signed commits/tags, reviews).

Appendix A — GitFlow (Optional)

Branches
• main: production history.
• develop: integration branch for next release.
• feature/*: from develop; merge back to develop.
• release/*: stabilize; then merge to main + develop; tag.

• hotfix/*: from main; merge to main + develop; tag.

Use GitFlow when: long- lived releases, strict QA gates, or offline vendors.

Appendix B — Useful Commands

Setup & hygiene

```
git config --global pull.ff only
git config --global init.defaultBranch main
git config --global commit.gpgsign true
```

Typical cycle

```
git checkout main && git pull --ff-only
git checkout -b feature/xyz
# work …
git add -A && git commit -m "feat(scope): message"
git fetch origin && git rebase origin/main
git push -u origin feature/xyz
```

Tagging

```
git tag -a v1.2.3 -m "Release v1.2.3"
git push origin v1.2.3
```

Revert

```
git revert <commit-sha>  # creates a new commit that reverts changes
```