

SoloDB release cheat sheet

Features, release candidates, production hotfixes and changes of plan

main	develop	release/X.Y
Production history	Shared integration testing	Selected work for one release

1 Build and test a feature

```
main -> feature/<ticket>-<description>
```

Start from current **main**. Implement and test. When ready for shared integration testing, open a PR from the feature into **develop**. Use a merge commit and keep the feature branch until it ships or is cancelled.

2 Select a feature for a release

```
same feature branch -> release/X.Y
```

Open a second PR from the **same feature branch** into the chosen release. Include version, dependencies, release notes, test evidence and deployment/schema steps. Merge after the release manager confirms scope. Integration into develop is useful, but optional.

3 Start a release

```
latest main -> release/X.Y
```

Create the release branch from **main**. Record version/date, selected feature PRs, dependencies, database/rollout work and the release decision owner. Merge only selected features. Release trains ship in version order; sync earlier releases from main before accepting later ones.

4 Fix a release-candidate bug

```
release/X.Y -> bugfix/<ticket>-<description>
```

Fix on a branch from the affected release; PR back into that release. Also apply the correction to develop where relevant. After feature freeze, accept only fixes, documentation, release notes and required deployment changes. Test the **complete release branch** in staging.

Before release acceptance

PHP lint, static analysis and PHPUnit; TypeScript build and relevant user flows; Doctrine schema/database review; intended SOLODB_HOST profiles; generated template maps, release notes and rollback instructions. CI and deployment must cover release branches.

```
Tag after the accepted PR merges into main
git switch main
git pull --ff-only origin main
git tag -a v5.15.0 -m "SoloDB 5.15.0"
git push origin v5.15.0
```

Cleanup: delete the release branch after verified deployment and completed follow-up. Delete feature branches after shipment and once all target PRs are closed.

5 Urgent production hotfix

```
main -> hotfix/<version>-<description>
```

Start from current **main**. PR into main with normal checks and focused regression tests. Tag the merge commit with the **next patch version**, deploy and verify. Forward-sync main through PRs into develop and **all active release branches**.

6 Publish a release

```
accepted release/X.Y -> main
```

Complete acceptance and final review; merge the release PR with a **merge commit**. Annotate and push a version tag on the resulting main commit. Deploy the tag, publish notes and verify production. Forward-sync main through PRs into develop and every later active release.

7 Postpone an unmerged feature

```
open release PR -> new target release PR
```

Close the existing release PR and open one from the same feature branch into the new target release. Record dependencies; a dependent feature cannot ship before its prerequisite.

8 Postpone an already merged feature

```
reviewed revert -> sync -> restore
```

Revert the feature merge on the original release through a reviewed PR. After that release reaches main, sync main into the later release and restore the feature with a reviewed **revert-of-revert or corrective PR**. Re-merging the original feature branch alone will not restore it.

Rules for every scenario

Every merge: PR, review and passing checks. Never merge develop wholesale into main or a release. Use merge commits for feature/release PRs; do not rebase shared branches or push to protected branches. Prefer forward merges from main; cherry-picks are for recovery. **Tags are immutable.**

Example: release **v5.15.0**;
next hotfix **v5.15.1**.
Deploy the tag.