Dependency management

- Releases
  - Decision to upgrade
    - Development branch
    - Stable branch
  - Release notes
  - Reviewing renovate pull requests
  - Tuning a software upgrade
- Soft forks
  - Temporary

Forgejo relies on hundreds of Free Software components and they all need to be updated on a regular basis, with appropriate tooling and methods.

Releases

Software referenced by a release (even if such a release is the hash of a commit). They are listed in the dependency dashboard which is updated by renovate from the renovate.json configuration file.

Decision to upgrade

Development branch

An upgrade is justified if:

- it is beneficial to Forgejo
- the risk of regression is low compared to the benefit

There is no need to upgrade if there is no indication that it is beneficial to Forgejo.

Stable branch

The dependency dashboard can be used to create a pull request for an upgrade on a stable branch. It is not done automatically.

An upgrade is justified if it follows the restrictions that apply to stable branches (not backporting features for instance).

Release notes

When an upgrade has a user facing impact on Forgejo, it should be included in the release notes.

- Label the pull request with worth a release note
- Add a `release-notes/<pr number>.md` file that contains one line for each feature or bug fix. For instance:
  - `feat!: [new feature](https://example.com/pull_request) with breaking changes`
  - `feat: [new feature](https://example.com/pull_request)`
  - `fix!: [bug fix](https://example.com/pull_request) with breaking changes`
  - `fix: [bug fix](https://example.com/pull_request)`

See this pull request for an example.

Reviewing renovate pull requests

Pull requests are opened when an upgrade is available and the decision to merge (positive review) or not (request for change review) depends on what the upgrade offers. The history of past upgrades can be browsed by looking for PR with the same title (e.g. happy-dom upgrades).

- The PR contains information about the release. If it does not, it has detailed references that can be used to browse the commits in the dependency source repository and figure out what the changes are.
- The comment of the review:
  - explains the decision (needed, not needed)
  - explains why the change has an impact on Forgejo
- If the upgrade is needed, user visible changes must be included in the draft release notes for the upcoming release. See this upgrade for an example.
- Security fix and important bug fixes are backported to the stable releases.

Note that renovate will keep a few (see `prConcurrentLimit` in renovate.json) pull request open at any given time. If no decision is made, newer upgrades will accumulate in the backlog visible in the dashboard.

The release team looks after the pull requests, to the extent that they can be tested automatically. If manual testing is required (because there is no test coverage for the part of the code that would be impacted by an upgrade), a review will be requested from the people who have the required expertise to either improve the test coverage or come up with a manual test procedure to be repeated.

Tuning a software upgrade

There is no uniformity in how software is released and they call for different strategies to deal with upgrades:

- **grouping related software**.
  When the decision to upgrade applies to a number of related software, it is less noisy to have them all upgraded in a single PR rather than a number of individual PRs. Such dependencies can be grouped together.
  - **using a renovate group preset:** e.g. `group:linters` include `eslint`, `eslint-plugin-array-func`, `eslint-plugin-github` etc. See also an example PR.
  - **creating a new group:**

```
{
  "description": "Group golang packages",
  "matchDepNames": [
"go",
"golang",
"docker.io/golang",
"docker.io/library/golang"
  ],
  "groupName": "golang packages"
},
```

- **release on every commit or so**.

  There are usually no release notes and there is no notion of release (monaco-editor) which may lead to frequent proposals to upgrade. It is similar to software that it tagged with a commit hash instead of a version, either because it does not publish versions (go-ap) or because a particular bug fix is needed before the release is available (go-rpmutils).
  - control the upgrade frequency with `schedule` (e.g. `schedule:quarterly` for pprof).
  - impose a delay with `minimumReleaseAge` (e.g. `monaco-editor` upgrades are considered no more frequently than once a month).
  - require dashboard approval with `dependencyDashboardApproval` (e.g. `go-ap` upgrades will never be proposed unless manually required from the dashboard.

- **automerge CI dependencies**.
  The dependencies that are exclusively used in the CI and demonstrated to work as expected when it passes can be merged automatically. They are listed in renovate.json) in the `Automerge some packages when CI succeeds` stanza as follows.
  - **extends:** if the software is included in a known renovate package preset (e.g. "packages:linters"). Figuring out if that is the case requires looking at the output of a renovate run and analyzing the debug logs.
  - **matchDepNames:** to explicitly list the dependency (e.g. `markdownlint-cli`).
  - **matchPackagePrefixes:** if a range of CI related dependency happen to share the same prefix (e.g. `@playwright/`)
- **automerge patch releases**.
  When a software is known to be good at publishing quality patch releases (in the semver sense), the proposed upgrades can be merged automatically. This can be done in a way similar to `vue` in renovate.json).

```
{
  "matchDepNames": [
□"vue"
  ],
  "separateMinorPatch": true
},
{
  "matchDepNames": ["vue"],
  "matchUpdateTypes": ["patch"],
  "automerge": true
},
```

Soft forks

Temporary

- https://code.forgejo.org/forgejo/download-artifact
- https://code.forgejo.org/forgejo/upload-artifact

□□□ #2
□ □□□ □□□ 2026-10-31 13:34:46 UTC
□ □□□ □□□ 2026-10-31 13:34:47 UTC