<epam> | CD/CD foundation for JavaScript engineers

RELEASE AND VERSIONING AUTOMATION TOOLS FOR JS ENGINEERS

EPAM Systems Inc.

Learn & Development
CD/CD foundation for JavaScript engineers
www.epam.com

RELEASE AND VERSIONING AUTOMATION TOOLS

The main goal of this article is to show the role of versioning in release flow, and possibility to use automated tools in this regard. We will review real life examples as well, which hopefully will give you more options to choose from in future. But the final decision is always yours, depending on the specific context: project itself, team, existing processes in the organization etc.

RELEASE VERSIONING CONTEXT

There are a lot of definitions of the release in software development. For the simplicity sake let's stuck with this one:

> Release is delivering ready **product** to the final **user.**

Where the **product** could be anything: web or mobile **app**, **library**, **microservice** etc. And the **user** can be different as well: it can be real **clients** or other **programmers**.

This differentiation leads to the different contexts of the release versioning. Versioning in one or another way is preferably applicable for any serious product (see the reasoning in the examples below). Versioning convention also may play a crucial role for the product's users. Or it may not, depending on the context. Let's review some context examples from real life:

1. **Product**: open source npm library

   Users: programmers

   Versioning purpose: indicate public API changes to the library users, so they can adjust their project's code accordingly

Versioning convention importance: crucial, without it users will not be able to use the library in case of incompatible change.

Versioning convention: most certainly semver (recommended by npm team)

2. **Product**: account management hub (Angular SPA), without ability/need for the users to see the app version

   Users: clients

   Versioning purpose: contribution to clearness of delivery process (e.g., integration with Jira to keep track of tasks in a release), simplifying management of different app states (e.g., for roll-back to previous working version in case of a disaster; canary releases; a/b testing etc.)

   Versioning convention importance: medium, not necessary for the end app users, but good to have for the smooth and clear delivery process.

   Versioning convention: could be semver (familiar to most developers, has tools for automation), or custom release names (e.g., release date, just major version update for rare release cadence etc.).

3. **Product**: mobile game, app version is exposed to the users

   Users: clients

   Versioning purpose: notify end users about the update to continue using the app, contribution to clearness of delivery process

   Versioning convention importance: medium – it should be at least unique and incremented on a new version publish, so the users can update the game (manually or by auto update). Visually the version does not really matter for the end users, they can notice it on some occasion, e.g., when they face with some issue and need to report it to support with providing the app's version.

Versioning convention: could be semver, or some hybrid of it (e.g., management doesn't want real users to see ugly versions like 26.12.6, but rather to keep the first number low and update second number on both major and minor changes, and the last number for the bugfixes, e.g. 1.12.2; or another variant - the build number can be included in the version, e.g. 1.12.2.001).

It's only a 3 use cases, but we can see how the context matters, so at the end use your best judgment and team/customer agreement to decide what release strategy and versioning convention suits best for the need.

AUTOMATION TOOLS FOR RELEASE

Preparation of release among other things usually requires following steps:

- Bump version in package.json / other version tracking source
- Update changelog with release notes
- Commit changelog and package*.json files changes to git
- Create git tag

These actions can be automated, and we can use this free time for something more exiting.

If you choose semver as a versioning convention, there is an eco-system build around it already:

- Conventional commit – convention for commits format (e.g. "feat: [User Profile] Add ability to store password")
- Standard version – library for release automation
- Semantic release – library for release automation, alternative to Standard version lib if you are aiming completely automating your release process as an output from CI/CD

Release flow can also be different depending on selected release strategy, but it's a topic for another time (leaving this here just in case).

AUTOMATED RELEASE FLOW EXAMPLE

Now let's focus on the automation of the above steps. We'll use Standard version library for that. Note: standard version only affects local git repo, it doesn't push anything to remote. Examples are the best, so let's dive into the release flow of the app AAA from the start to the end.

Context: app AAA has git flow release strategy (separate release branch created from develop for each release), release cadences is each 2 weeks, Conventional commit is followed by developers, Standard version package is installed, current app version is 1.3.6.

Workflow for AAA project sprint X goes like this:

1. Developer 1 has done next tasks:

```
feat: [AAA-Rewards]  Add possibility to share reward rate with friends

fix: [AAA-Rewards] Invalid reward amount for friend invite
```

   Developer 2 has done next task:

```
feat: [AAA-UserAccount]  Add possibility to update profile photo
```

2. Sprint comes to end, release is coming. By this time we already have clean commit history in git thanks to Conventional commit (and probably commitlint to check commit format along the way).

3. All we need to do now is to run `standard-version` command (more detailed tutorial here).

This will automatically bump app version in package*.json files, update release notes in CHANGELOG.md and commit these changes. After app version will be changed from 1.3.6 to 1.4.0 (according to semver – no breaking changes were introduced, only backward compatible new functionality and bug fixes). CHANGELOG.md will contain new section like this:

```
1.4.0  (2021-09-01)

Features

[AAA-Rewards]  Add possibility to share reward rate with friends (7563c63)

[AAA-UserAccount]  Add possibility to update profile photo (7f55f5e)

Bug Fixes

[AAA-Rewards] Invalid reward amount for friend invite (#3252) (7f55f5e)
```

And a commit with a similar message will be created:

```
chore(release): 1.4.0
```

That's it. We have a ready-to-deploy app version with all needed release artifacts locally. Now we can push the changes to remote repo. After that, based on project's CI/CD configuration and other project's release aspects, the changes can go through quality gates, be deployed to lower environments for smoke and/or performance testing, etc. And eventually be deployed to prod env for the end users.