Cours Logiciels Libres

Produire des nouvelles versions d'un logiciel libre

Théo Zimmermann

Télécom Paris, Institut Polytechnique de Paris

Vendredi 24 février 2023

À quoi ça sert ?

Si le code est déjà disponible sur un **dépôt public** et que n'importe qui peut le télécharger, le compiler, l'installer, à quoi peut-il servir de produire des **versions numérotées** ?

À quoi ça sert ?

Si le code est déjà disponible sur un **dépôt public** et que n'importe qui peut le télécharger, le compiler, l'installer, à quoi peut-il servir de produire des **versions numérotées** ?

- Désigner une version du code **recommandée** pour les utilisateurs (ou sur laquelle des retours sont attendus).
- Faire connaître les **changements** récents.
- Rendre disponible à travers des **gestionnaires de paquets** :
  - **Gestionnaire de paquets spécifique** (à un écosystème donné : npm, pip, opam, etc.) $\rightarrow$ mis à jour par les mainteneurs du logiciel.
  - **Gestionnaire de paquets du système** (distributions Linux, Homebrew, etc.) $\rightarrow$ mis à jour par des contributeurs indépendants.
- Produire des installateurs / exécutables.

Le contenu d'une “release”

Essentiel :

Le contenu d'une “release”

Essentiel :

- Un **tag** dans un dépôt public (généralement associé à une **archive** compressée du code source—que GitHub génère automatiquement).

Le contenu d'une “release”

Essentiel :

- Un **tag** dans un dépôt public (généralement associé à une **archive** compressée du code source—que GitHub génère automatiquement).

Recommandé :

Le contenu d'une “release”

Essentiel :

- Un **tag** dans un dépôt public (généralement associé à une **archive** compressée du code source—que GitHub génère automatiquement).

Recommandé :

- Des **notes de versions** détaillant la liste des changements.

Le contenu d'une “release”

Essentiel :

- Un **tag** dans un dépôt public (généralement associé à une **archive** compressée du code source—que GitHub génère automatiquement).

Recommandé :

- Des **notes de versions** détaillant la liste des changements.

Optionnel :

Le contenu d'une “release”

Essentiel :

- Un **tag** dans un dépôt public (généralement associé à une **archive** compressée du code source—que GitHub génère automatiquement).

Recommandé :

- Des **notes de versions** détaillant la liste des changements.

Optionnel :

- **Paquets** dans un ou plusieurs gestionnaires de paquets.
- **Exécutables** précompilés / installateurs.

La numérotation des versions

- Les versions sont généralement **numérotées** (parfois elles peuvent avoir des noms de code en plus).
- Un numéro de version est composé d'un ou **plusieurs composants** (le plus souvent deux ou trois) séparés par des points. Exemple : 3.12.7.
- Le point n'est pas un séparateur décimal : après 1.9, il peut y avoir 1.10.
- Les numéros sont **incrémentés** avec le temps (mais plusieurs séries peuvent être maintenues en parallèle).
- Une version préliminaire peut être indiquée par un **suffixe** comme "beta" ou "rc" (release candidate). Exemple : 1.0-rc1 (première release candidate de la version 1.0).

Signification des numéros de version

Il existe deux approches communes :

Signification des numéros de version

Il existe deux approches communes :

1. Le **versionnement sémantique** (*semver*)

   - Premier composant : numéro de version **majeur**.
   - Deuxième composant : numéro de version **mineur**.
   - Troisième composant : numéro de version de **correctif**.

   Règles :

   - Un changement casse la **compatibilité arrière**
     $\rightarrow$ on incrémente le numéro **majeur**.
   - Un changement casse la **compatibilité avant** mais pas la compatibilité arrière (nouvelles **fonctionnalités**)
     $\rightarrow$ on incrémente le numéro **mineur**.
   - Un changement ne casse ni la compatibilité arrière ni la compatibilité avant (**correctifs** de bug)
     $\rightarrow$ on incrémente le numéro de **correctif**.
   - Dans les versions **0.x**, tout est permis (le logiciel n'est considéré **stable** qu'à partir de la version 1.0).

Signification des numéros de version

2. Le versionnement calendaire (*calver*).

   Exemples : Ubuntu 20.04, 21.04, 21.09 (YY.MM).

   Parfois une combinaison de calendrier et de sémantique :

   - (YY)YY.MINOR.PATCH
   - (YY)YY.MM.MINOR

Notes de version

Pourquoi faire ?

Notes de version

Pourquoi faire ?

- Pour les **utilisateurs** :

Notes de version

Pourquoi faire ?

- Pour les **utilisateurs** :
  - information sur les changements qui cassent la **compatibilité**,
  - **découverte** des ajouts et des correctifs,
  - permet de décider quand **mettre à jour**.

Notes de version

Pourquoi faire ?

- Pour les **utilisateurs** :
  - information sur les changements qui cassent la **compatibilité**,
  - **découverte** des ajouts et des correctifs,
  - permet de décider quand **mettre à jour**.
- Pour les **mainteneurs** / utilisateurs expérimentés :

Notes de version

Pourquoi faire ?

- Pour les **utilisateurs** :
  - information sur les changements qui cassent la **compatibilité**,
  - **découverte** des ajouts et des correctifs,
  - permet de décider quand **mettre à jour**.
- Pour les **mainteneurs** / utilisateurs expérimentés :
  - une référence de **quand** les changements ont été introduits,
  - pour déterminer si un changement est normal ou **inattendu**.

Standard “Keep a Changelog” https://keepachangelog.com/

- Un fichier `CHANGELOG.md`.
- Avec les notes de version dans l'**ordre chronologique inversé**.
- Peut commencer par une section `Unreleased`.
- Pour chaque version : **numéro et date**.
- Description des changements regroupés en **catégories** :
  - `Added` : **ajouts** de fonctionnalités.
  - `Changed` : modifications (impact possible sur la **compatibilité**).
  - `Deprecated` : annonce des fonctionnalités **bientôt supprimées**.
  - `Removed` : fonctionnalités **désormais supprimées**.
  - `Fixed` : corrections de **bugs**.
  - `Security` : corrections de **vulnérabilités**.

GitHub Releases

- Attention : **en dehors du dépôt versionné** / des sources.
- Met en avant la dernière version sur la page web du dépôt. Option pour **s'abonner** aux nouvelles versions (*watch*).
- Création du **tag** (*optionnel*) sur une branche au choix. **Génération automatique des notes** de version (*optionnel*) :
  - listent les changements basée sur les **pull requests**,
  - possibilité d'auto-exclusion / catégorisation basée sur les labels,
  - peuvent être ensuite éditées manuellement,
  - **créditent les contributeurs** de la nouvelle version.
- Dépôt de fichiers **attachés** : exécutables, documentation, etc.
- Peut être générée / mise à jour par un workflow CI / CD.
- Option pour marquer une version comme version préliminaire.
- **Utilisation recommandée**, en *complément* d'un `CHANGELOG.md`.

Gestion des branches et sortie de versions

- Le plus simple si vous êtes seul mainteneur : pousser les changements et les **tags sur la branche principale** (*main* / *master*).
- Problème si plusieurs mainteneurs : le travail de sortie de nouvelle version peut bloquer d'autres changements.
  $\rightarrow$ On utilise une **branche** pour **stabiliser** la nouvelle version.
- L'utilisation de plusieurs branches permet aussi de sortir plusieurs versions en parallèle (exemple : 4.0.0, 3.10.0, 3.9.3).

Portage arrière

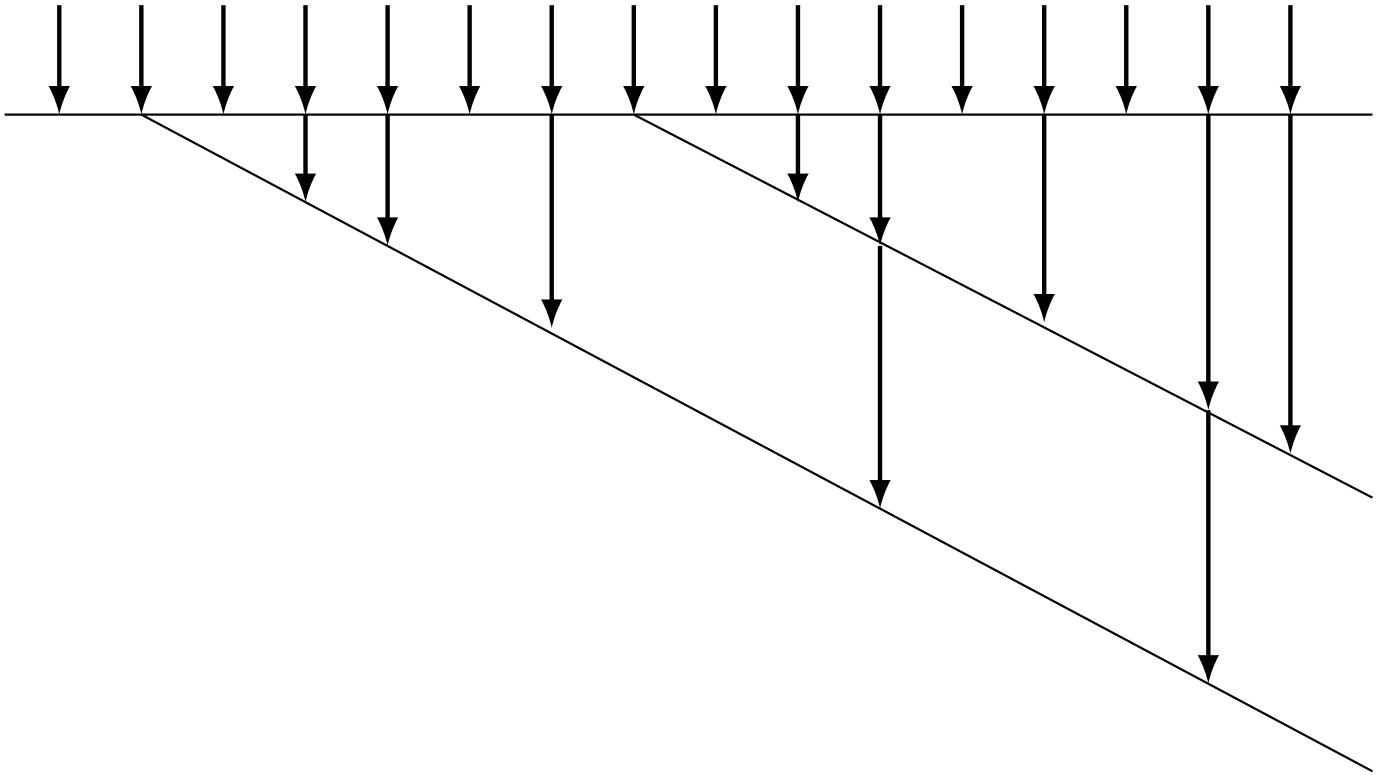

Figure 1: Flot des modifications : les *pull requests* sont intégrées dans la branche principale, avant qu'une sélection soit appliquée (*backport*) sur les branches stables qui sont activement maintenues.

Stabilisation / test d'une nouvelle version

- Créer une nouvelle **branche** pour stabiliser la nouvelle version.
- Les développeurs et certains utilisateurs testent la **version de développement** $\rightarrow$ on peut corriger certains problèmes connus avant toute sortie de version.

Stabilisation / test d'une nouvelle version

- Créer une nouvelle **branche** pour stabiliser la nouvelle version.
- Les développeurs et certains utilisateurs testent la **version de développement** $\rightarrow$ on peut corriger certains problèmes connus avant toute sortie de version.
- Sortir et annoncer une **version préliminaire** (beta1, rc1) pour que davantage d'utilisateurs la testent.

Stabilisation / test d'une nouvelle version

- Créer une nouvelle **branche** pour stabiliser la nouvelle version.
- Les développeurs et certains utilisateurs testent la **version de développement** $\rightarrow$ on peut corriger certains problèmes connus avant toute sortie de version.
- Sortir et annoncer une **version préliminaire** (beta1, rc1) pour que davantage d'utilisateurs la testent.
- **Corriger les bugs** importants qui ont été découverts et sortir une nouvelle version préliminaire ou une version finale.

Stabilisation / test d'une nouvelle version

- Créer une nouvelle **branche** pour stabiliser la nouvelle version.
- Les développeurs et certains utilisateurs testent la **version de développement** $\rightarrow$ on peut corriger certains problèmes connus avant toute sortie de version.
- Sortir et annoncer une **version préliminaire** (beta1, rc1) pour que davantage d'utilisateurs la testent.
- **Corriger les bugs** importants qui ont été découverts et sortir une nouvelle version préliminaire ou une version finale.
- Qui décide de ce qui est porté sur la branche stable ?

Stabilisation / test d'une nouvelle version

- Créer une nouvelle **branche** pour stabiliser la nouvelle version.
- Les développeurs et certains utilisateurs testent la **version de développement** $\rightarrow$ on peut corriger certains problèmes connus avant toute sortie de version.
- Sortir et annoncer une **version préliminaire** (beta1, rc1) pour que davantage d'utilisateurs la testent.
- **Corriger les bugs** importants qui ont été découverts et sortir une nouvelle version préliminaire ou une version finale.
- Qui décide de ce qui est porté sur la branche stable ?
  - *Release manager*.
  - Travail d'équipe (vote / règles strictes pour l'inclusion).

  Dans tous les cas : faire attention à quel changement inclure.

Cycles de production de versions

Deux alternatives :

Cycles de production de versions

Deux alternatives :

- Des cycles basés sur les **fonctionnalités** (le plus commun pour des projets nouveaux ou avec de petites équipes).

Cycles de production de versions

Deux alternatives :

- Des cycles basés sur les **fonctionnalités** (le plus commun pour des projets nouveaux ou avec de petites équipes).
- Ou basés sur le **calendrier** (beaucoup de projets libres finissent par basculer vers cette option, cf. thèse de Martin Michlmayr).

Cycles de production de versions

Deux alternatives :

- Des cycles basés sur les **fonctionnalités** (le plus commun pour des projets nouveaux ou avec de petites équipes).
- Ou basés sur le **calendrier** (beaucoup de projets libres finissent par basculer vers cette option, cf. thèse de Martin Michlmayr).

  Différentes durées de cycle sont possibles :

Cycles de production de versions

Deux alternatives :

- Des cycles basés sur les **fonctionnalités** (le plus commun pour des projets nouveaux ou avec de petites équipes).
- Ou basés sur le **calendrier** (beaucoup de projets libres finissent par basculer vers cette option, cf. thèse de Martin Michlmayr).

  Différentes durées de cycle sont possibles :

  - Une version (importante) tous les six mois. Example : Ubuntu.
  - Une version toutes les six semaines (**cycles rapides**). Exemple : Chrome.

Cycles de production de versions

Deux alternatives :

- Des cycles basés sur les **fonctionnalités** (le plus commun pour des projets nouveaux ou avec de petites équipes).
- Ou basés sur le **calendrier** (beaucoup de projets libres finissent par basculer vers cette option, cf. thèse de Martin Michlmayr).

  Différentes durées de cycle sont possibles :

  - Une version (importante) tous les six mois. Example : Ubuntu.
  - Une version toutes les six semaines (**cycles rapides**). Exemple : Chrome.

  Implique de :

Cycles de production de versions

Deux alternatives :

- Des cycles basés sur les **fonctionnalités** (le plus commun pour des projets nouveaux ou avec de petites équipes).
- Ou basés sur le **calendrier** (beaucoup de projets libres finissent par basculer vers cette option, cf. thèse de Martin Michlmayr).

  Différentes durées de cycle sont possibles :

  - Une version (importante) tous les six mois. Example : Ubuntu.
  - Une version toutes les six semaines (**cycles rapides**). Exemple : Chrome.

  Implique de :

  - Maintenir la branche principale dans un état convenable :
    - inclure la documentation avec les ajouts dans la même PR,
    - ne jamais casser les tests (avec de l'intégration continue).
  - Repousser les fonctionnalités qui ne sont pas prêtes.