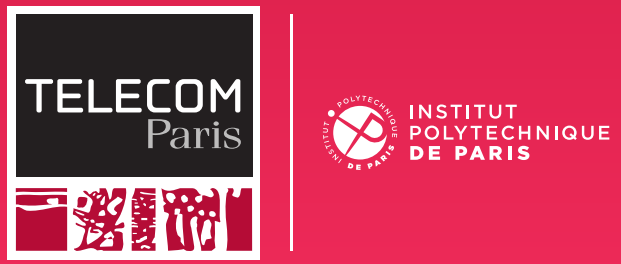
3TC37: Contribution à un logiciel libre

Communication

Marc Jeanmougin
Théo Zimmermann

Télécom Paris, Institut Polytechnique de Paris



Communiquer en public

La plupart des discussions dans un projet se font en public.

- Permet à chacun d'avoir la réponse à une question que potentiellement plusieurs se posent
- Y compris des non-contributeurs ou nouveaux arrivants
- Permet la tracabilité sur les prises de décisions

Hors des discussions sur les problèmes interpersonnels, il y a peu de bonnes raisons de "cacher" des discussions avec des informations concernant le projet.

https://producingoss.com/en/producingoss.html#written-culture

Synchrone vs asynchrone

- Synchrone : chats, discord, visio, événements physiques, etc.
- Asynchrone : mailing-lists, forums, wiki, issue tracker, MR/PR, etc.

Dans les deux cas, respecter le code of conduct et le temps de ses interlocuteurs.

Quels sont les avantages et inconvénients des discussions synchrones ? asynchrones ?

Poser de bonnes questions

- Lisez la documentation (*RTFM*)
- Essayez de trouver la réponse par **vous-même** *avant* de poster la question
- **Mettez en doute** les réponses obtenues avec une IA *avant* de poster la question
- Si vous posez une question, dites ce que vous avez essayé et où vous avez cherché la réponse - en détail si possible
- Don’t ask to ask
- Adoptez les coutumes locales (dire bonjour, dire si vous avez finalement trouvé la réponse par vous-même (et la donner), etc.)
- Votre situation n’intéresse en général pas les gens (“Je suis étudiant et …”, “J’ai un problème urgentissime :”, etc.)

Refs: http://www.catb.org/~esr/faqs/smart-questions.html (daté), https://jvns.ca/blog/good-questions/

Forges et gestionnaires de version

SourceForge

By Slashdot Media LLC, Public Domain

- Lancé en 1999, c'est l'une des premières **forge** de logiciels libres.
- Permet aux projets d'avoir un dépôt de code, un bug tracker, un wiki, une mailing list ou un forum, une page de téléchargement, etc.
- Simplifie le partage de **petits projets open source**, les petits forks de projets non maintenus, etc.
- Plus de 500 000 projets y ont été hébergés.
- Mais en 2013, le site a perdu en crédibilité à cause de certaines pratiques douteuses, et beaucoup de projets ont alors migré vers GitHub.

git

By Jason Long, CC BY 3.0

- Les gestionnaires de versions traditionnels (CVS, SVN) étaient **centralisés** (nécessitent un dépôt central, un accès à ce dépôt, voir l'utilisation de verrous) : peu adaptés au modèle de **collaboration ouverte** de l'open source.
- Le projet Linux bascule sur un gestionnaire **décentralisé mais propriétaire** : BitKeeper. La licence permet de l'utiliser pour développer des logiciels libres, mais son utilisation provoque une controverse et la licence finit par être révoquée.
- Linus Torvalds crée **git** à partir de 2005 comme alternative libre à BitKeeper.

GitHub

By GitHub, Public Domain

- **GitHub est lancé en 2008** par quatre développeurs en Californie.
- Forge basée sur git.
- Popularise le concept de **pull request** et l'utilisation de **forks de développement**.
- Devient rapidement le site de référence pour les projets open source (également très utilisé par les entreprises pour héberger leurs projets internes) : plus de 200 millions de projets aujourd'hui.

Les écosystèmes de paquets

- Les débuts (avant les forges) : des archives pour **partager des extensions** (Emacs Lisp Archives, CTAN, CPAN, CRAN, etc.)
- Les **distributions Linux** se construisent autour de leur gestionnaire de paquet (dpkg en 1994 pour Debian, qui sert toujours de brique de base pour APT).
- Apparitions des **écosystèmes de paquets** modernes construits autour de gestionnaires de paquets spécifiques dans les années 2000 (RubyGems, Cabal) et 2010 (npm, pip, Composer, opam, Cargo, etc.)
- Explosion du nombre de **dépendances** des projets logiciels. Conséquences médiatiques quand des tout petits paquets disparaissent (LeftPad).

Processus de contribution

Avant de contribuer

Commencer par **comprendre et respecter** les règles (écrites et non écrites).

- Lire la documentation : licence, `README`, guide de contribution, code de conduite, politique IA, etc.
- Se familiariser avec le projet (naviguer dans le système de tickets, sur les forums, les chats, s'inscrire à l'activité).

- Règles génériques : être poli, **ne pas s'impatienter**, répondre aux questions calmement…
- Communiquer tôt, communiquer fréquemment : avant d'implémenter, discutez (sur un chat, sur le rapport de bug) - surtout avant un gros changement qui prendra du temps

Rapporter un bug

Si vous découvrez un bug en utilisant un logiciel libre, vous pouvez le rapporter aux mainteneurs du logiciel.

Rapporter un bug

Si vous découvrez un bug en utilisant un logiciel libre, vous pouvez le rapporter aux mainteneurs du logiciel.

1. Chercher le système de tickets / *bug tracker*.

Rapporter un bug

Si vous découvrez un bug en utilisant un logiciel libre, vous pouvez le rapporter aux mainteneurs du logiciel.

1. Chercher le système de tickets / *bug tracker*.
2. Chercher avec différents mots-clés si le bug a déjà été rapporté. Si c'est le cas, vous pouvez éventuellement y ajouter des informations nouvelles qui aident à le reproduire ou débugger.

Rapporter un bug

Si vous découvrez un bug en utilisant un logiciel libre, vous pouvez le rapporter aux mainteneurs du logiciel.

1. Chercher le système de tickets / *bug tracker*.
2. Chercher avec différents mots-clés si le bug a déjà été rapporté. Si c'est le cas, vous pouvez éventuellement y ajouter des informations nouvelles qui aident à le reproduire ou débugger.
3. Si vous n'avez pas trouvé de ticket similaire, vous pouvez en créer un nouveau. Pensez à bien indiquer toute information utile pour reproduire le bug :
   - **version** du logiciel,
   - comment vous avez obtenu / compilé le paquet,
   - quel est votre **environnement** (système d'exploitation, navigateur…),
   - détails sur comment **reproduire le problème** (court et complet).

   N'utilisez surtout pas l'IA pour écrire le rapport de bug ! (~~Copilot~~)

Rapporter un bug

Si vous découvrez un bug en utilisant un logiciel libre, vous pouvez le rapporter aux mainteneurs du logiciel.

1. Chercher le système de tickets / *bug tracker*.
2. Chercher avec différents mots-clés si le bug a déjà été rapporté. Si c'est le cas, vous pouvez éventuellement y ajouter des informations nouvelles qui aident à le reproduire ou débugger.
3. Si vous n'avez pas trouvé de ticket similaire, vous pouvez en créer un nouveau. Pensez à bien indiquer toute information utile pour reproduire le bug :
   - **version** du logiciel,
   - comment vous avez obtenu / compilé le paquet,
   - quel est votre **environnement** (système d'exploitation, navigateur…),
   - détails sur comment **reproduire le problème** (court et complet).

   N'utilisez surtout pas l'IA pour écrire le rapport de bug ! (~~Copilot~~)
4. Pensez à répondre aux questions des mainteneurs.

Reproduire un bug

- Parfois, les **rapports de bug s'accumulent** sur de longues périodes.
- Parfois, il **manque des informations utiles** pour le débuggage.
- Vous pouvez aider en essayant de **reproduire** des bugs déjà rapportés.
  - Si oui, en expliquant dans **quel environnement** et avec **quelle version du logiciel** vous avez réussi à reproduire le bug,
  - Précisez **comment reproduire le bug** si les instructions initiales sont insuffisantes ou si vous les avez simplifiées.
  - Si non, cela peut éventuellement signifier que le **bug a été corrigé** depuis ou qu'il est lié à un **environnement particulier**.
- Vérifier dans le guide de contribution que ce genre de contribution est utile et acceptée.

Contribuer un changement

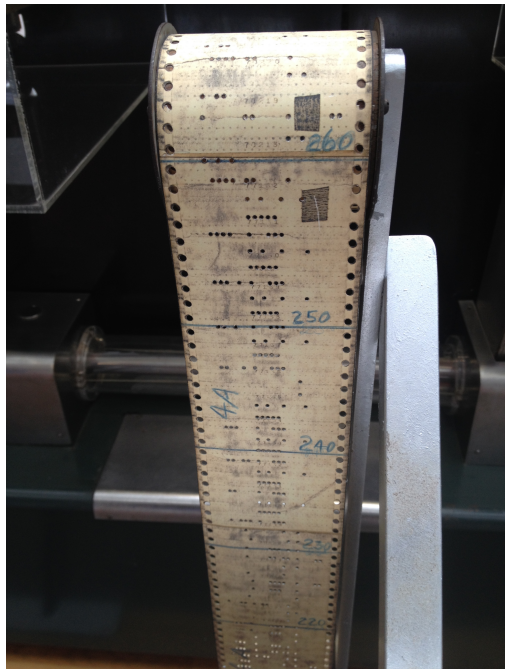
Rappel : penser à **communiquer** avant de se lancer dans le code.

Pour soumettre un patch, cela dépend du projet :

- Patch envoyé par e-mail (cf. https://git-send-email.io/). Exemple : Linux.
- Patch en pièce attachée sur un bug tracker.
- **Merge request** sur GitLab / GitHub (le plus commun pour les projets récents, mais aussi certains projets anciens).

Pour savoir : trouver et **lire la documentation** (“contributing guide”).

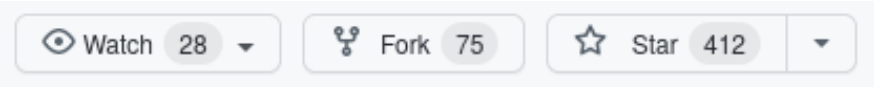
Les patchs



Le mot patch trouverait son origine dans la manière dont on corrigeait les bugs dans les programmes sur ruban perforé.

Les pull requests / merge requests

1. Cliquer sur le bouton “Fork” sur GitHub / GitLab.



2. Cloner le dépôt (votre fork).

```
$ git clone git@github.com:mon-compte/le-projet
```

3. Créer une nouvelle branche (**important !**) :

```
$ git switch --create ma-branche
```

Pourquoi ?

Les pull requests / merge requests

4. Coder… (il peut être utile de **lire la documentation**, de poser des questions sur le ticket si besoin, etc.)
5. Lancer les tests en local si possible.
6. Commit (voir la doc sur le format de message) et push :

```
$ git add mes fichiers modifies
$ git commit -m "Mon message de commit en 50 caractères max.

Des informations supplémentaires sur plusieurs lignes,
après un saut de ligne."
$ git push -u origin ma-branche
```

Les pull requests / merge requests

7. Ouvrir la pull request (par exemple en cliquant sur le lien que la forge logicielle donne au moment de pousser, ou en naviguant sur le dépôt du projet).
8. Écrire un message expliquant les modifications et les tickets qu'elle ferme.

Les pull requests / merge requests

9. Prendre en compte les retours sur la pull request :
   - Retours automatiques de l'intégration continue et de bots éventuels.
   - Commentaires des mainteneurs.

   (Si besoin) modifier le code dans la branche, commit, push.

   Selon les attentes du projet, ça peut devenir plus compliqué et apprendre à bien maitriser git peut être utile (cf. https://git-rebase.io/).

Quelques écueils à éviter

- Ne pas **être disponible pour répondre** aux questions, prendre en compte les suggestions.
- Les **trop gros** changements : plus votre contribution est grosse, plus la revue de code est difficile, et donc plus elle prendra du temps à être intégrée.
- Mélanger des changements **indépendants** : les mainteneurs préfèrent éviter cela en général. De plus, plusieurs contributions plus petites sont plus rapides à relire qu'une seule trop grosse.
- Les fonctionnalités dont la **conception** n'est pas arrêtée. Il vaut mieux mélanger le moins possible revue du design et revue du code. Pour discuter du design, **un ticket suffit**.

“AI slop” et politiques IA

Consignes pour le TP :

- par groupes de 3 ou 4, sélectionner un des liens vers une discussion / une politique IA de projets open source proposées sur Grist,
- assignez-vous les liens à l’aide de la colonne idoine du Grist,
- lisez la discussion, et préparez une synthèse à présenter à la classe (en 3 minutes environ) : quels sont les points de vue exprimés, les arguments, les exemples, et les conclusions (en termes de politique IA).