INF113: Contribution à un logiciel libre

Introduction, théorie et principes du logiciel libre

Marc Jeanmougin et Théo Zimmermann

Télécom Paris, Institut Polytechnique de Paris

Mercredi 12 février 2025

Section 1

Introduction

Objectifs du cours

- Développement de vos compétences en programmation (`INF11[1-3]`)

  Notamment en apprennant à :

  - naviguer dans du code existant,
  - planifier une modification,
  - prendre en compte les retours automatiques et les revues de code,
  - justifier de choix techniques.

- Connaissance des principes et des enjeux du logiciel libre :

  - définition et notions historiques,
  - compréhension et respect des licences,
  - capacité à évaluer un projet avant de l'utiliser ou d'y contribuer.

Modalités pratiques

Cours très orienté vers la pratique. Objectif : contribuer à un logiciel libre.

Séances :

- Théorie & principes, histoire, licences
- Modèles de gouvernance, communautés, modèles économiques
- TP préparatoires au projet (recherche de projets, analyse de projets)
- TP d'aide au projet (fin mars/début avril)
- Contrôle de connaissance (09/04)
- Présentations (28/04)

Évaluation :

- Point d'étapes notés, écrits ou oraux (30%)
- Contrôle de connaissance (30%)
- Présentation finale du projet (40%)

Qui sommes-nous ?

- Marc Jeanmougin, co-fondateur du MOOC Open Source Masterclass, co-mainteneur du logiciel libre Inkscape.
- Théo Zimmermann, enseignant-chercheur sur le(s) modèle(s) de développement collaboratif du logiciel libre, co-mainteneur du logiciel libre Coq.

C'est quoi un logiciel libre ? Un logiciel propriétaire ?

C'est quoi un logiciel libre ? Un logiciel propriétaire ?

Un *logiciel libre*, c'est un logiciel distribué sous une *licence* qui autorise ceux qui le reçoivent à l'**utiliser**, en **étudier le code source**, le **modifier** et le **redistribuer** (inchangé ou modifié).

Le contraire d'un logiciel libre est un *logiciel propriétaire*.

Il s'agit donc de garantir des *libertés* à l'utilisateur (là où les licences de logiciels propriétaires restreignent l'utilisation, l'accès au code source et la redistribution).

Mais au-delà de la question des libertés, cela permet aussi un autre modèle de développement, collaboratif et ouvert, qu'on désigne plus souvent sous le terme d'*open source*.

Vous utilisez déjà des logiciels libres

Avec vos voisins, établissez une liste de :

- logiciels libres que vous utilisez,
- logiciels propriétaires que vous utilisez.

Vous utilisez déjà des logiciels libres

Avec vos voisins, établissez une liste de :

- logiciels libres que vous utilisez,
- logiciels propriétaires que vous utilisez.

Directement :

- Votre système d'exploitation si vous êtes sous GNU/Linux,
- Votre navigateur web si c'est Firefox ou Chromium,
- Une partie de votre téléphone si vous êtes sous Android,
- Votre éditeur de texte si c'est Emacs, Vim, VS Code ou Atom,
- Le compilateur de votre langage préféré,
- Le gestionnaire de version git,
- etc.

Indirectement (dépendances à des bibliothèques libres) :

- La plupart des sites web sur lesquels vous naviguez,
- La plupart des applications mobiles que vous utilisez.

Pourquoi ce cours est utile pour vous

- En tant que futur · es ingénieur · es.
- En tant que futur · es chercheur · ses.
- En tant que citoyen · nes.
- À titre personnel.

En tant que futur · es ingénieur · es

En tant que futur · es ingénieur · es

- Pour vous faire recruter : votre profil GitHub est votre deuxième CV.
- L'**utilisation de logiciel libre** est essentielle dans presque toutes les entreprises (dans et hors secteur informatique) :
  - Réduction des coûts,
  - Adaptabilité,
  - Durabilité.
- Le logiciel libre peut aussi être moteur de **collaborations** au-delà des frontières de l'entreprise.

En tant que futur · es chercheur · ses

En tant que futur · es chercheur · ses

Trois piliers de la **science ouverte** : open access, open data, open source

- Partage de code source en open source $\Longrightarrow$
  - reproduire les résultats de la recherche,
  - construire à partir des résultats existants,
  - comparer différentes techniques.
- Opportunité pour créer des communautés internationales de chercheur · ses autour de projets (exemple : Coq, Scikit-learn, etc.).

En tant que citoyen · nes

En tant que citoyen · nes

- Loi pour une République numérique (2016) : les administrations doivent publier leurs codes et leurs données.
  - https://code.gouv.fr/ (2021)
- On peut inspecter le code source du calcul des impôts, de ParcourSup, TousAntiCovid, etc. (mais ces logiciels ne sont pas nécessairement développés de manière ouverte).
- Des collectifs développent du logiciel libre au service de la démocratie participative : voir Code for France, Decidim, etc.

À titre personnel

À titre personnel

Lorsque vous **utilisez** des logiciels libres, vous pouvez :

- en rapporter des bugs,
- regarder et modifier le code source pour corriger des bugs, l'adapter à vos besoins,
- faire partie d'une communauté.

Si vous **développez** un logiciel ou une bibliothèque et que vous souhaitez le partager et recevoir des contributions.

Section 2

Définition et histoire

Free software, logiciel libre et open source

Ces trois termes veulent fondamentalement dire la même chose, mais en mettant l'accent sur différents aspects.

Free software, logiciel libre et open source

- Le terme “free software” a été popularisé par Richard Stallman et la Free Software Foundation (1985).
- Il met l’accent sur les **libertés** (“Free Software Definition”) :
  0. Liberté d’**utiliser** le programme, pour n’importe quel usage.
  1. Liberté d’**étudier** comment le programme fonctionne et de le **modifier** (requiert l’accès aux sources).
  2. Liberté de **redistribuer des copies** du programme.
  3. Liberté de **redistribuer des versions modifiées** du programme.
- Son problème est que “free” peut vouloir dire “libre” ou “gratuit”. L’expression *“Free as in free speech, not as in free beer”* rappelle que ce n’est pas une question de prix.

Free software, **logiciel libre** et open source

- Le terme “logiciel libre” (“software libre” en espagnol) est la traduction de “free software”.
- Il n’a pas ce problème de double sens.
- Certaines personnes choisissent de l’utiliser en anglais pour éviter la confusion ($\rightarrow$ LibreOffice).

Free software, logiciel libre et **open source**

- Le terme “open source” a été popularisé par Eric Raymond et l’Open Source Initiative (1998).
- Il met l’accent sur le **partage** du code permettant la **collaboration ouverte** (“Open Source Definition”).
- Son problème est que beaucoup de gens confondent “publication du code source” et “open source”, alors que le code source peut être publié sous une licence restrictive. De nos jours, de nombreuses entreprises et organisations publient du code sans qu’il soit libre / open source pour autant.
- À l’heure des grands modèles d’IA, certains sont tentés d’abuser du terme open source pour désigner certains modèles en accès ouvert, mais cela soulève de nombreuses questions.

Exercice : des modèles d'IA open source ?

Chercher des informations sur internet sur les questions que soulève l'usage du terme "open source" pour désigner des modèles d'IA.

Free software, logiciel libre et open source

Pour éviter toute confusion et éviter de prendre parti dans un débat philosophique, certains parlent de FOSS ou FLOSS :
pour **F**ree, (**L**ibre,) and **O**pen **S**ource **S**oftware.

En pratique, la quasi-totalité des logiciels libres sont open source et inversement.

Travail à faire pour la semaine prochaine

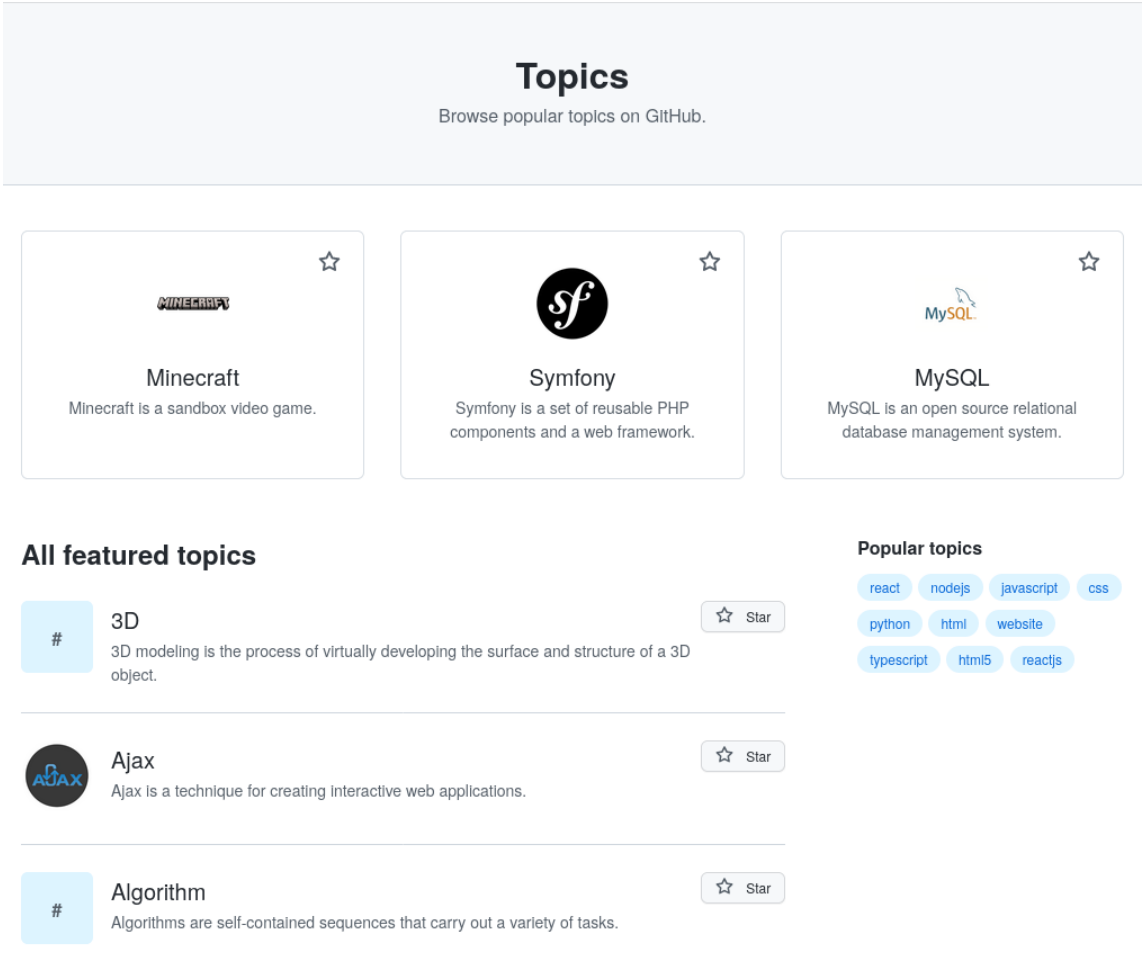
- Trouvez :
  - 2 logiciels libres utilisant **un langage de programmation / une technologie que vous maitrisez**,
  - 2 logiciels libres qui vous **intéressent personnellement** mais dans d'**autres langages de programmation** que les précédents (que vous maîtrisez ou non).
- Pour chacun, ouvrir une **issue** dans le dépôt inf113-docs en remplissant le template.

Pour vous aider à trouver des projets

Annuaire du Libre de l'association Framasoft : https://framalibre.org

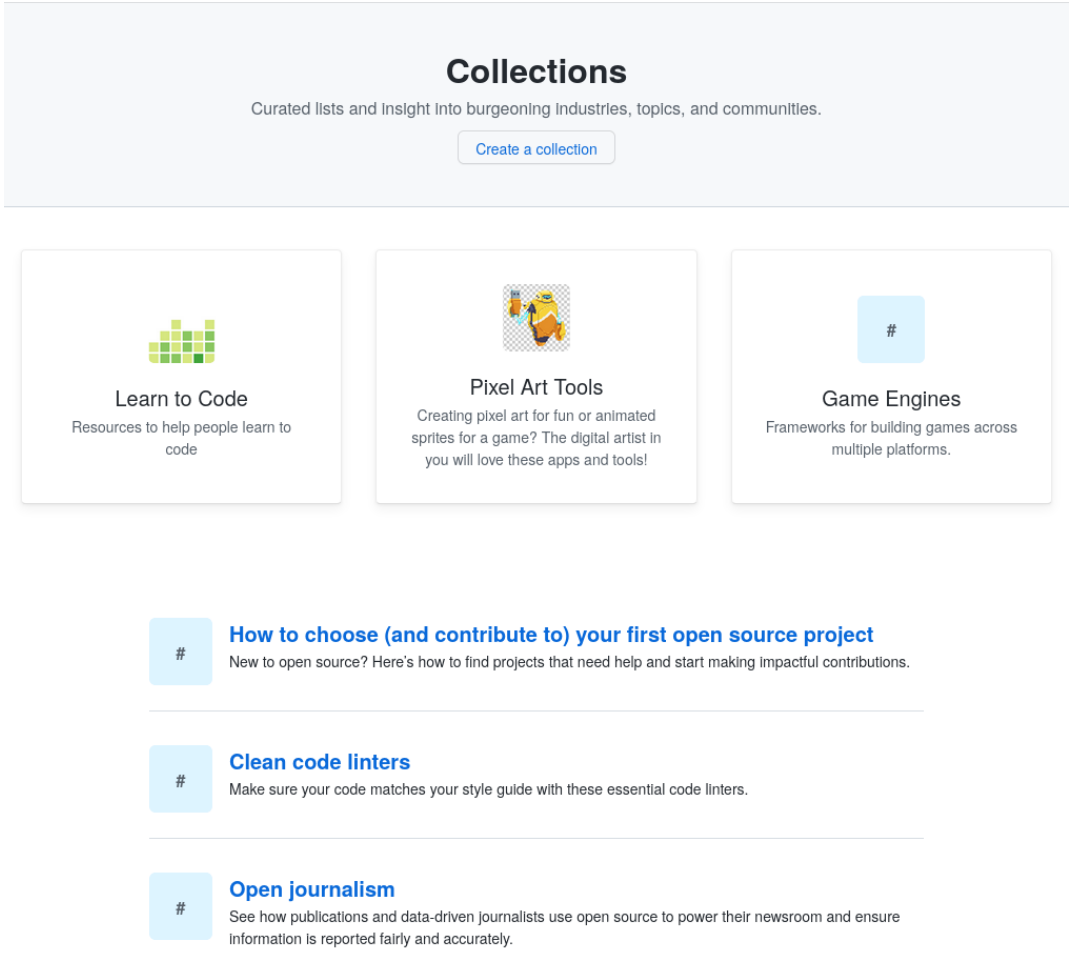
Pour vous aider à trouver des projets

GitHub Topics : https://github.com/topics



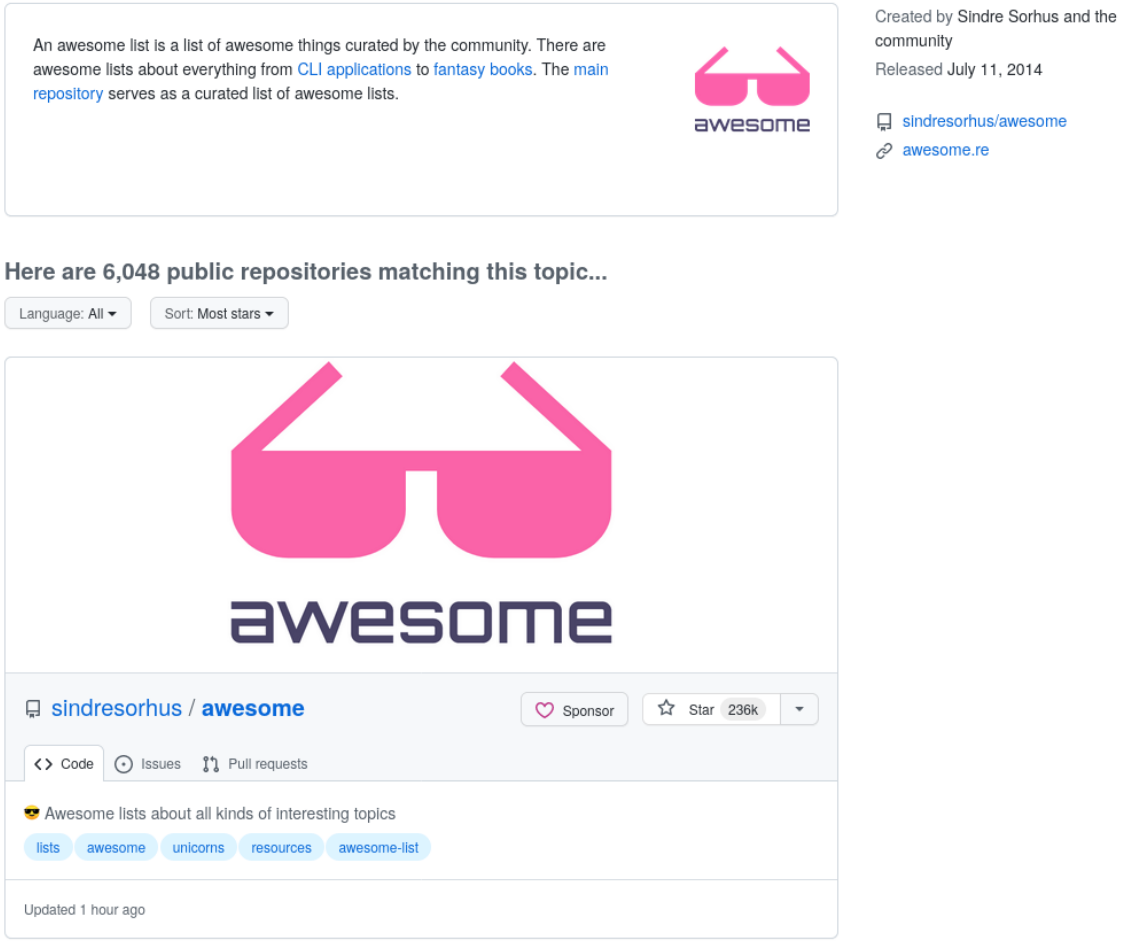
Pour vous aider à trouver des projets

GitHub Collections : https://github.com/collections



Pour vous aider à trouver des projets

Awesome Lists : https://github.com/sindresorhus/awesome



Histoire du logiciel libre

- De quand date le logiciel libre ?
- Mais en fait, de quand date le logiciel ?
- Quel âge ont certains logiciels libres célèbres ?

Avant le logiciel libre

- Dans les débuts de l'informatique, **les ordinateurs coûtent très cher** et sont distribués avec des logiciels spécialement conçus pour eux (dont la valeur est secondaire).
- Des programmes sont modifiés, conçus et partagés par leurs utilisateurs, sans règles particulières.
- On finit par considérer que le logiciel a une valeur en soit, peut être vendu et est protégé par le **copyright** (droit d'auteur).
- **Microsoft prend son envol** à travers un accord avec IBM : IBM vend des PC (personal computers) avec le système d'exploitation de Microsoft (MS-DOS), sur lequel ce dernier conserve les droits. L'industrie du logiciel est née.

Walter Isaacson

L'auteur de la biographie de référence de Steve Jobs

Les Innovateurs

Comment un groupe de génies,
hackers et geeks a fait la révolution numérique

9.

Le logiciel

Lorsque Paul Allen s'approcha du kiosque à journaux encombré au milieu de Harvard Square et aperçut l'Altair en couverture du *Popular Electronics* daté de janvier 1975, il fut à la fois enthousiasmé et consterné. Bien qu'il soit emballé par la révélation que l'ère de l'ordinateur individuel avait commencé, il craignait d'arriver trop tard pour participer à la fête. Il posa rageusement ses soixante-quinze *cents* sur le comptoir, s'empara du magazine et trotta dans la neige fondue jusqu'à la chambre en cité universitaire de Bill Gates, son pote de lycée de Seattle, mordu d'informatique lui aussi, qui l'avait convaincu de décrocher de la fac et de s'installer à Cambridge. « Hé ! ce truc est en train de nous passer sous le nez ! », déclara Allen. Gates se mit à osciller d'avant en arrière, comme cela lui arrivait souvent dans des moments particulièrement intenses. Quand il eut fini de lire l'article, il se rendit compte qu'Allen avait raison. Pendant les huit semaines suivantes, ils se lancèrent dans une frénésie programmatrice qui allait changer la nature commerciale de l'informatique[1].

Figure 2: Les Innovateurs, Chapitre 9

Subsection 1

Les premiers logiciels libres

Unix (pas un logiciel libre)

- Lancé à partir de 1969 chez AT&T (Bell Labs).
- Distribué à de nombreux industriels et universités qui en créent des **variants**.
- Aujourd'hui les descendants (spirituels) d'Unix sont :
  - La famille BSD (Berkeley Software Distribution) :
    - FreeBSD, OpenBSD, NetBSD,
    - Darwin, le noyau (libre) de macOS.
    - À l'origine de la famille de **licences BSD**.
  - GNU / Linux (GNU = GNU's Not Unix)

Le projet GNU

Figure 3: By Aurelio A. Heckert, CC BY-SA 2.0

- Lancé à partir de 1983 par Richard Stallman.
- Pour créer un Unix complètement libre.
- Une **collection de projets** divers : GNU Emacs, GCC, glibc et GDB, GNU Bash, GNU Core Utilities (cat, ls, rm, etc.), GRUB, GParted, GIMP, GPG, etc.
- Et la famille de **licences GNU** (GPL, LGPL, AGPL).
- Création de la Free Software Foundation en 1985.

X Window System (X11)

Figure 4: By Sven, CC BY-SA 3.0

- Protocole de système de fenêtrage (encore) très utilisé sous Linux.
- Lancé à partir de 1984 au MIT.
- Distribué sous licence X11, à l'origine de la **licence MIT**.
- Implémentation actuelle X.Org, depuis 2004 par la X.Org Foundation.

Le noyau Linux

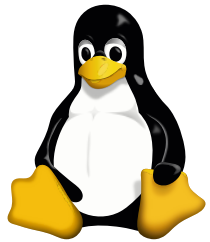


Figure 5: By Larry Ewing, Simon Budig, Garrett LeSage, CC0

- Lancé à partir de 1991 par Linus Torvalds.
- Premier logiciel libre **développé de manière collaborative** sur internet (envoi de patchs par newsgroups, puis par e-mail).
- Sous **licence GNU GPL 2.0** depuis 1992.
- A inspiré l'essai "The Cathedral and the Bazaar" à Eric Raymond.
- Création de la **Linux Foundation** en 2000, qui collecte aujourd'hui plus de 15 millions de dollars annuels de la part de ses plus de 1000 entreprises membres.

Les premières distributions GNU/Linux

- Debian :
  - Lancée à partir de 1993 par Ian Murdock.
  - L'une des toutes premières distributions Linux
  - Entièrement **communautaire** et presque entièrement libre.
- Red Hat (distribution et entreprise) :
  - Distribution lancée à partir de 1994 par Marc Ewing.
  - L'une des toutes premières entreprises à produire une distribution Linux et en **vendre la maintenance**.
  - Version communautaire : Fedora.
  - L'entreprise Red Hat est **rachetée par IBM** en 2019 pour 34 milliards de dollars.

Section 3

Licences

Aspects juridiques et licences : pourquoi c'est important

- Les licences de logiciels déterminent ce que vous avez le **droit** ou non de faire avec.
- Votre entreprise encourt des **risques économiques** à ne pas respecter les termes des licences.
- À titre individuel, si vous ne courez pas de tel risques, il y a tout de même de bonnes raisons de prêter attention aux licences :
  - Une raison **morale** : le logiciel libre est une forme de dons sous conditions. Acceptez les conditions ou refusez le don.
  - Une raison **pratique** : vous pouvez mettre en danger vos utilisateurs et certains refuseront donc d'utiliser vos logiciels si vos pratiques de gestion des licences laissent à désirer.

Subsection 1

Propriété intellectuelle et logiciels

La propriété intellectuelle en bref

La propriété intellectuelle est composée de plusieurs champs, dont les trois suivants :

- Le **droit d'auteur** (**copyright** dans les pays anglo-saxons).
- Les **brevets**.
- Le droit des **marques**.

Ces trois domaines peuvent concerner le **logiciel**.

Note : le terme "propriété intellectuelle" est controversé (car très loin de la notion de propriété sur des objets physiques). Mais il est pratique pour regrouper les différentes notions sous-jacentes.

Le droit d'auteur

- Protège des **œuvres originales de l'esprit**, indépendamment de leur support.
- Protège l'**expression** (créative) et non les idées.
- S'applique automatiquement, sans besoin d'enregistrement préalable.
- Deux parties :
  - **droit moral** (notamment reconnaissance de la paternité de l'œuvre) inaliénable.
  - **droits patrimoniaux** (monopole d'exploitation pour durée limitée, par exemple 70 ans après la mort de l'auteur) qui peuvent être cédés.
- Correspond au **copyright** dans les pays anglo-saxons (sans droit moral).
- Portée internationale : Convention de Berne (1886, adoptée aux USA en 1989).

Les brevets

- Ils permettent d'obtenir un **monopole** sur l'exploitation d'inventions industrielles.
- Ce monopole est à **durée limitée** (généralement 20 ans) et nécessite un **dépôt** (de brevet).
- L'obtention d'un brevet nécessite de prouver que l'idée est **nouvelle**, **utile** et **non-évidente**.
- Le principe des brevets est basé sur un compromis censé favoriser l'**innovation** :
  - Le dépôt de brevet nécessite de rendre l'invention **publique**.
  - En échange, on accorde un monopole temporaire à son inventeur.
  - Mais au terme de cette durée, n'importe qui peut se servir de l'invention.

Les marques

- Noms, slogans, logos, etc.
- Elles permettent de protéger le consommateur car elles indiquent la **provenance** d'un produit.
- Elles ne nécessitent pas toujours un enregistrement, mais doivent être **activement utilisées** et sont limitées à un **marché spécifique**.

Application au logiciel

- Copyright / droit d'auteur : depuis les années 70.
  - Plus simple à appliquer (pas besoin d'enregistrement).
  - Protège les logiciels qui ne contiennent pas d'idées nouvelles.
  - S'applique aux codes **sources**, aux codes **binaires**/**objets**, aux éléments **graphiques**, et aussi à la **documentation**.
  - Règles spéciales pour les droits patrimoniaux du logiciel.
- Brevets :
  - Initialement considérés comme ne s'appliquant pas au logiciel.
  - Puis accordés de plus en plus facilement, aux USA et dans l'UE.
  - Peuvent empêcher les **clones** libres de logiciels propriétaires.
  - Générateurs de **risques juridiques**.
- Marques : peuvent s'appliquer aux noms et logos de logiciels, y compris de logiciels libres.

Subsection 2

Licences libres

Licences

- Une licence est un **contrat** qui permet au détenteur (du copyright, d'un brevet) d'accorder des droits au bénéficiaire de la licence.
- Tous les logiciels (libres et propriétaires) sont distribués avec des licences (**pas de licence = pas de droits**) :
  - Lorsque un logiciel propriétaire est vendu, les utilisateurs doivent "acheter" des licences.
  - Dans le cas du libre, les licences accordent des droits (qui peuvent être plus ou moins vastes) à tous ceux qui **reçoivent** le logiciel.
- Les licences logicielles reposent **principalement sur le droit d'auteur/copyright**, et secondairement sur les brevets (certaines licences libres modernes incluent des clauses sur les brevets).

Exemple de licence (MIT)

```
MIT License

Copyright (c) [year] [fullname]

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to
deal in the Software without restriction, including without limitation the
rights to use, copy, modify, merge, publish, distribute, sublicense, and/or
sell copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
IN THE SOFTWARE.
```

Conditions standards des licences libres

License	Commercial use	Distribution	Modification	Patent use	Private use	Disclose source	License and copyright notice	Network use is distribution	Same license	State changes	Liability	Trademark use	Warranty
BSD Zero Clause License	●	●	●		●						●		●
Academic Free License v3.0	●	●	●	●	●		●			●	●	●	●
GNU Affero General Public License v3.0	●	●	●	●	●	●	●	●	●	●	●		●
Apache License 2.0	●	●	●	●	●		●			●	●	●	●
Artistic License 2.0	●	●	●	●	●		●			●	●	●	●
BSD 2-Clause "Simplified" License	●	●	●		●		●				●		●
BSD 3-Clause Clear License	●	●	●	●	●		●				●		●
BSD 3-Clause "New" or "Revised" License	●	●	●		●		●				●		●
BSD 4-Clause "Original" or "Old" License	●	●	●		●		●				●		●
Boost Software License 1.0	●	●	●		●		●				●		●

Figure 6: Capture d'écran de https://choosealicense.com/appendix/

Conditions standards des licences libres

Toujours autorisés :

- **Usage** privé, usage commercial.
- **Modification**.
- **Distribution** (à l'identique, ou d'une version modifiée).

Brevets, 3 possibilités :

- Licence explicite (idéal).
- Absence de mention de la question des brevets (OK).
- Absence explicite de licence sur les brevets (mal).

Conditions possibles :

- **Paternité** / préservation de la licence / du copyright.
- Partage des sources, modifications sous la **même licence** (copyleft).
- Traçage des modifications (un dépôt git public suffit).

Licences libres les plus connues

Permissives :

- MIT.
- BSD3, BSD2.
- Apache 2.0 (explicite sur les brevets, traçage des changements).

Copyleft faible :

- MPL 2.0 (partage des modifications dans les mêmes fichiers).
- GNU LGPL 3.0, LGPL 2.1 (utilisation de bibliothèque).

Copyleft fort :

- GNU GPL 2.0, GPL 3.0 (ajouts en 3.0 : brevets, “tivoization”).
- GNU AGPL 3.0 (modification sur serveur $\implies$ distribution).

Licence copyleft (préambule de la GNU GPL 3.0)

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program--to make sure it remains free software for all its users. [..]

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Compatibilité entre licences

Les licences de deux logiciels libres sont compatibles si on peut créer un logiciel qui **dérive des deux à la fois**.

Exemple d'**incompatibilité** (entre licences copyleft) :

- La licence GNU GPL 2.0 (de Linux par exemple) impose que tout travail dérivé soit distribué sous licence GNU GPL.
- La licence CDDL (de Solaris) impose que tout travail dérivé soit distribué sous licence CDDL.

Compatibilité entre licences

Autre exemple d'incompatibilité (plus subtil) :

- La licence GNU GPL impose que tout travail dérivé soit distribué sous cette licence et empêche l'ajout de nouvelles restrictions.
- La licence BSD originale (4 clauses) contient une clause ajoutant des restrictions sur la publicité :

  All advertising materials mentioning features or use of this software must display the following acknowledgement: This product includes software developed by [project].

Compatibilité entre licences

La compatibilité avec les licences GNU est très utile, car les logiciels sous ces licences sont très répandus.

En pratique, de nombreuses licences ont été **modifiées suite à des problèmes d'incompatibilité** avec les licences GNU (en particulier la GPL) :

- BSD 2 et 3 clauses (version à 4 clauses incompatible),
- Apache 2.0 (version 1 incompatible),
- MPL 2.0 (version 1 incompatible),
- CeCILL 2.0 (version 1, B et C incompatibles),
- etc.

Les licences copyleft ont souvent besoin d'une clause spécifique autorisant l'**inclusion dans un logiciel GPL**.

Licences multiples

- On trouve parfois des logiciels qui sont distribués sous plusieurs licences. Dans ce cas, **l'utilisateur peut choisir** quelle licence s'applique.
- Certaines licences (notamment la GPL) peuvent être augmentées de **clauses additionnelles** (qui peuvent seulement accorder des droits supplémentaires, et peuvent être retirées) :
  - Classpath exception (Java)
  - LGPL 3.0
- L'auteur peut choisir d'autoriser l'utilisation de **nouvelles versions** d'une licence.
  - C'est une **bonne pratique** mais qui nécessite de faire confiance à l'organisme qui produit ces licences.
  - GPL-2.0-only (Linux) vs GPL-2.0-or-later.
  - Certaines licences le rendent automatique (exemple MPL 2.0).

Domaine public

Définition : une œuvre est dans le domaine public lorsqu'elle **n'est plus protégée** par des droits patrimoniaux.

"Mettre une œuvre dans le domaine public" : ce n'est **pas juridiquement possible** dans tous les pays. Par conséquent, des licences existent qui accordent des **droits équivalents** au cas où ce ne serait pas permis (Unlicense, CC-0).

Choisir une licence libre

Règle 1 : on n'oublie pas de mettre une licence sur son code !

Sans licence :

- Vos **utilisateurs** n'ont aucun droit.
- Vous n'avez aucun droit sur les **contributions externes** !

Choisir une licence libre

Règle 2 : on n'invente pas sa propre licence !

Conséquence de la prolifération de licences :

- Risques juridiques (licences moins comprises, moins testées).
- **Incompatibilité** entre licences (difficile de combiner des logiciels libres entre eux).

Il vaut mieux choisir une licence **populaire** :

- Soit une licence populaire en général (MIT, BSD2, BSD3, Apache 2.0, MPL 2.0, licences GNU).
- Soit une licence populaire dans l'**écosystème** auquel on contribue (mais en général elle fait partie de la liste précédente).

Choisir une licence libre

Règle 3 : on choisit une licence qu'on comprend !

- Certaines licences ont des conditions **complexes**. Par exemple la licence GNU LGPL donne des droits supplémentaires, mais à des conditions techniques qui ne sont pas autant adaptées à tous les langages de programmation.
- Il est **difficile de changer de licence** plus tard, une fois qu'on a reçu de nombreuses contributions externes.

Choisir une licence libre

- Domaine public ?
- Permissive ?
- Protection contre les brevets ?
- Copyleft faible ?
- Copyleft fort ?
- Le logiciel est-il destiné à tourner sur un serveur ?
- Modèle économique associé ?

Et pour la documentation ?

- Le plus commun est que la documentation soit sous la **même licence que le code**.
- Mais des licences spécifiques aux contenus autres que logiciels existent (par exemple, les licences **Creative Commons**).

Creative Commons

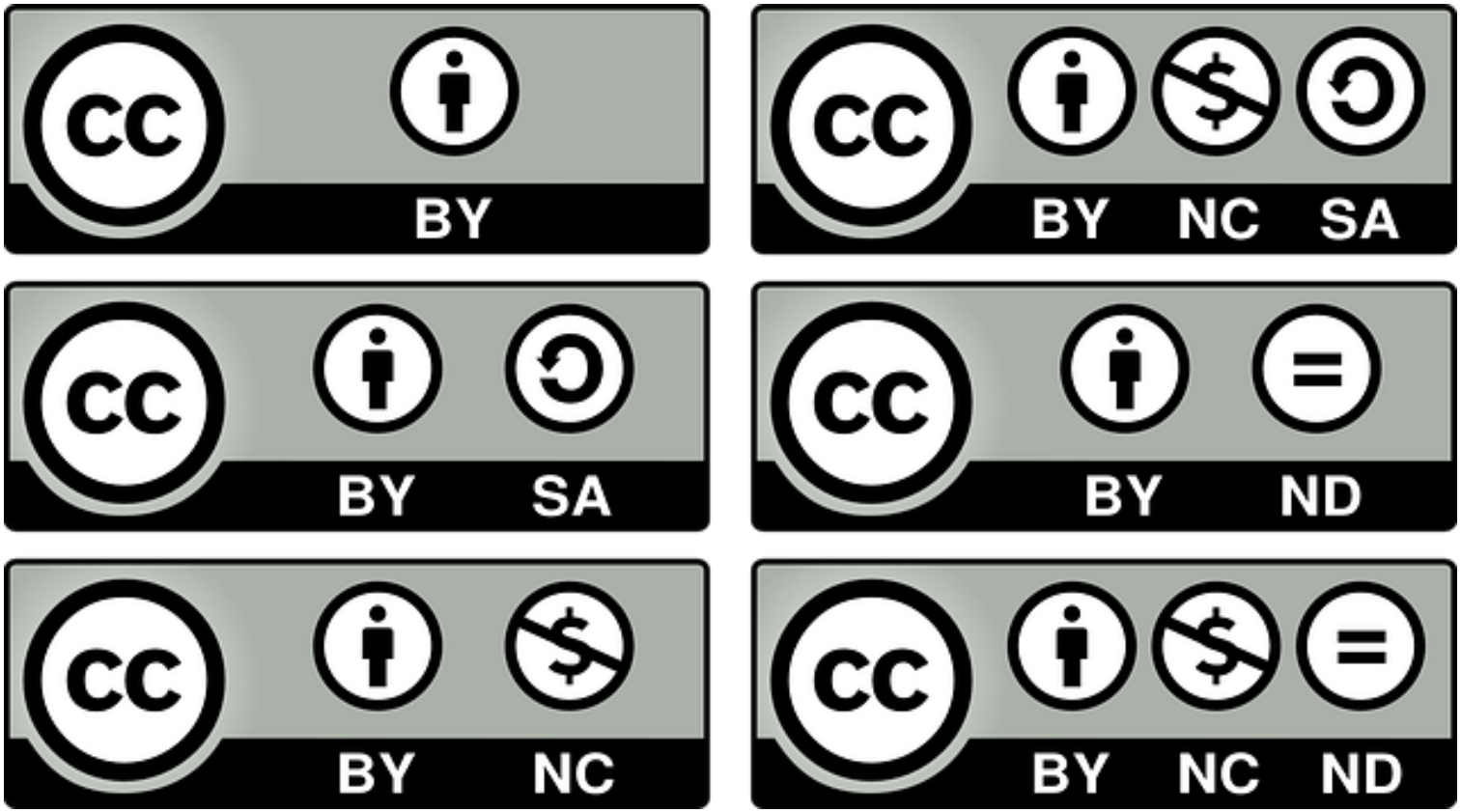


Figure 7: Les différentes licences Creative Commons

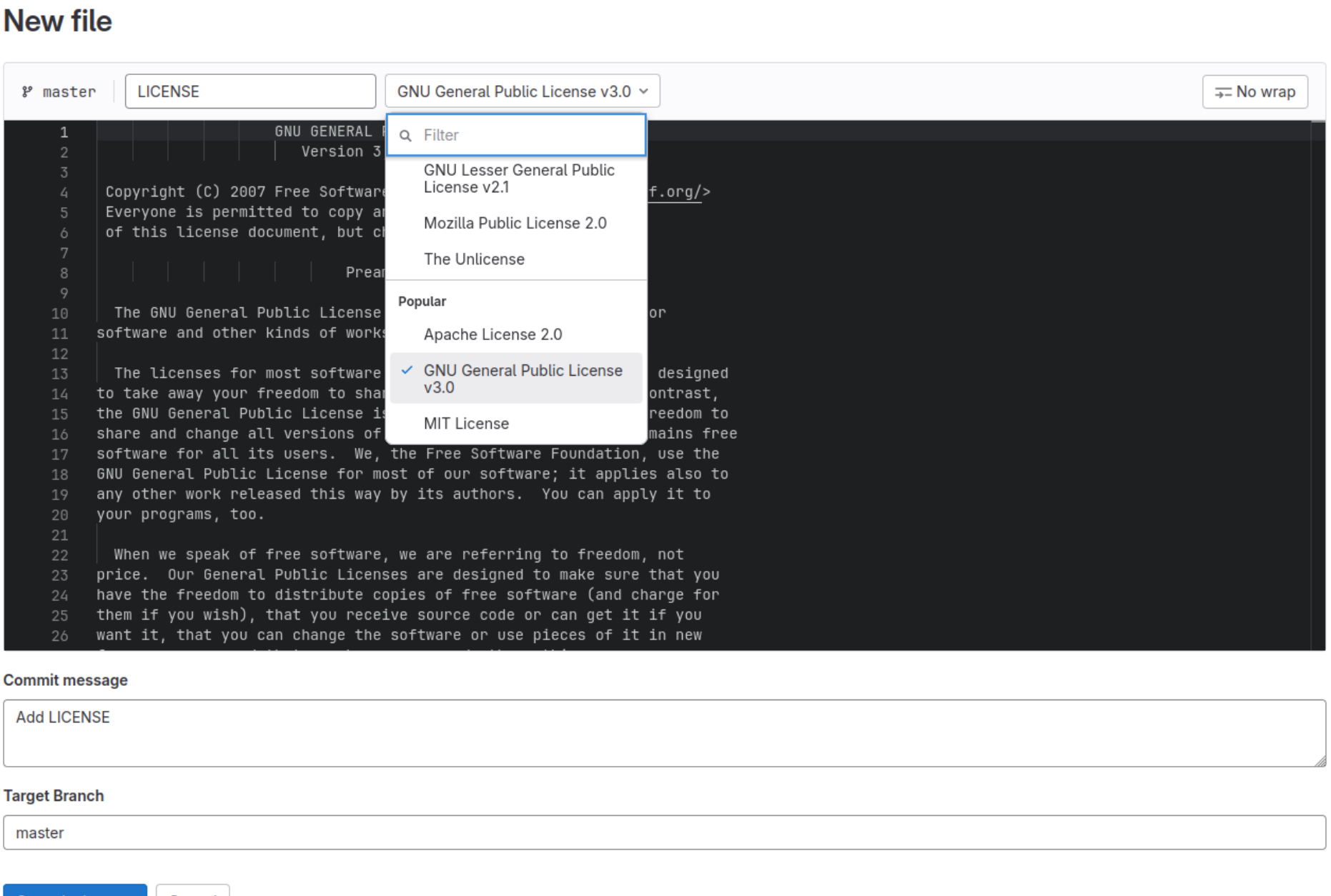
Attention ! Seules CC-BY et CC-BY-SA sont des licences libres.

Appliquer une licence libre

Les recommandations varient d'une licence à l'autre :

- En général, on inclut le **texte de la licence** dans un fichier `LICENSE` à la racine du projet.
- Certaines licences recommandent en plus l'ajout d'un **commentaire en haut de chaque fichier** de code source.
- Les fichiers de **métadonnées pour gestionnaires de paquets** (tel que `package.json` pour npm) ont souvent un champ pour indiquer la licence. On utilise alors en général la typologie **SPDX** pour ce champ.

Exemple : dans gitlab



Typologie SPDX https://spdx.org/licenses/

Licences les plus communes :

- MIT
- BSD-2-Clause, BSD-3-Clause
- Apache-2.0
- MPL-2.0
- LGPL-2.1-only, LGPL-2.1-or-later, LGPL-3.0-only, **LGPL-3.0-or-later**
- GPL-2.0-only, GPL-2.0-or-later, GPL-3.0-only, **GPL-3.0-or-later**
- AGPL-3.0-only, **AGPL-3.0-or-later**

Expressions :

- MIT **OR** Apache-2.0 (licences multiples)
- GPL-3.0-only **WITH** Classpath-exception-2.0 (clauses additionnelles)

Exemple

```
// SPDX-License-Identifier: GPL-2.0-or-later
/*
 * Editable view implementation
 *
 * Authors:
 *   Lauris Kaplinski <lauris@kaplinski.com>
 *   MenTaLguY <mental@rydia.net>
 *   bulia byak <buliabyak@users.sf.net>
 *   Ralf Stephan <ralf@ark.in-berlin.de>
 *   John Bintz <jcoswell@coswellproductions.org>
 *   Johan Engelen <j.b.c.engelen@ewi.utwente.nl>
 *   Jon A. Cruz <jon@joncruz.org>
 *   Abhishek Sharma
 *
 * Copyright (C) 2007 Jon A. Cruz
 * Copyright (C) 2006-2008 Johan Engelen
 * Copyright (C) 2006 John Bintz
 * Copyright (C) 2004 MenTaLguY
 * Copyright (C) 1999-2002 Lauris Kaplinski
 * Copyright (C) 2000-2001 Ximian, Inc.
 *
 * Released under GNU GPL v2+, read the file 'COPYING' for more information.
 */
```

Application de licences aux contributions

Certains projets demandent de “signer” des documents :

- DCO (Developer Certificate of Origin),
- CLA (Contributor License Agreement),
- CTA (Copyright Transfer Agreement).

Quelles peuvent être les raisons de ces demandes ?

DCO (Developer Certificate of Origin)

- Créé pour le projet Linux, également utilisé pour d'autres projets.
- Le contributeur s'engage sur l'origine de la contribution et sur son droit à la soumettre à la licence du projet.
- Très léger : nécessite une ligne par commit (Signed-off-by: Your Name <your@email.com>), option `-s` de git.

CLA (Contributor License Agreement)

Même utilité qu'un DCO, plus des droits supplémentaires (par exemple pour l'entreprise qui maintient le projet) :

- Par exemple, le droit d'inclure la contribution dans un code propriétaire.
- Ou bien le droit de changer la licence.

CTA (Copyright Transfer Agreement)

- Cession des droits patrimoniaux.
- Utilisé par la Free Software Foundation pour pouvoir lancer des procès au nom de tous les contributeurs.

Application de licences aux contributions

Extrait des Terms of Services de GitHub :

> *Whenever you add Content to a repository containing notice of a license,* ***you license that Content under the same terms, and you agree that you have the right to license that Content under those terms****. If you have a separate agreement to license that Content under different terms, such as a contributor license agreement, that agreement will supersede.*
> *Isn't this just how it works already? Yep. This is widely accepted as the norm in the open-source community; it's commonly referred to by the shorthand "inbound=outbound".* ***We're just making it explicit.***

Subsection 3

Licences non libres

Licences éthiques

Copyright (c) [year] [fullname]

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to [...], subject to the following conditions:

* The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

* The Software may not be used by individuals, corporations, governments, or other groups for systems or activities that actively and knowingly endanger, harm, or otherwise threaten the physical, mental, economic, or general well-being of individuals or groups in violation of the United Nations Universal Declaration of Human Rights (https://www.un.org/en/universal-declaration-human-rights/).

[...]

This license is derived from the MIT License, as amended to limit the impact of the unethical use of open source software.

Licences anti-concurrentielles

Exemple : Business Source License 1.1

- Source code **disponible**.
- **Usage** “en production” **restreint**.
  - Hashicorp (Terraform) ajoute une clause autorisant l’usage en production, sauf dans des produits concurrents.
- Les versions anciennes deviennent automatiquement open source après un certain temps.

Changements de licences

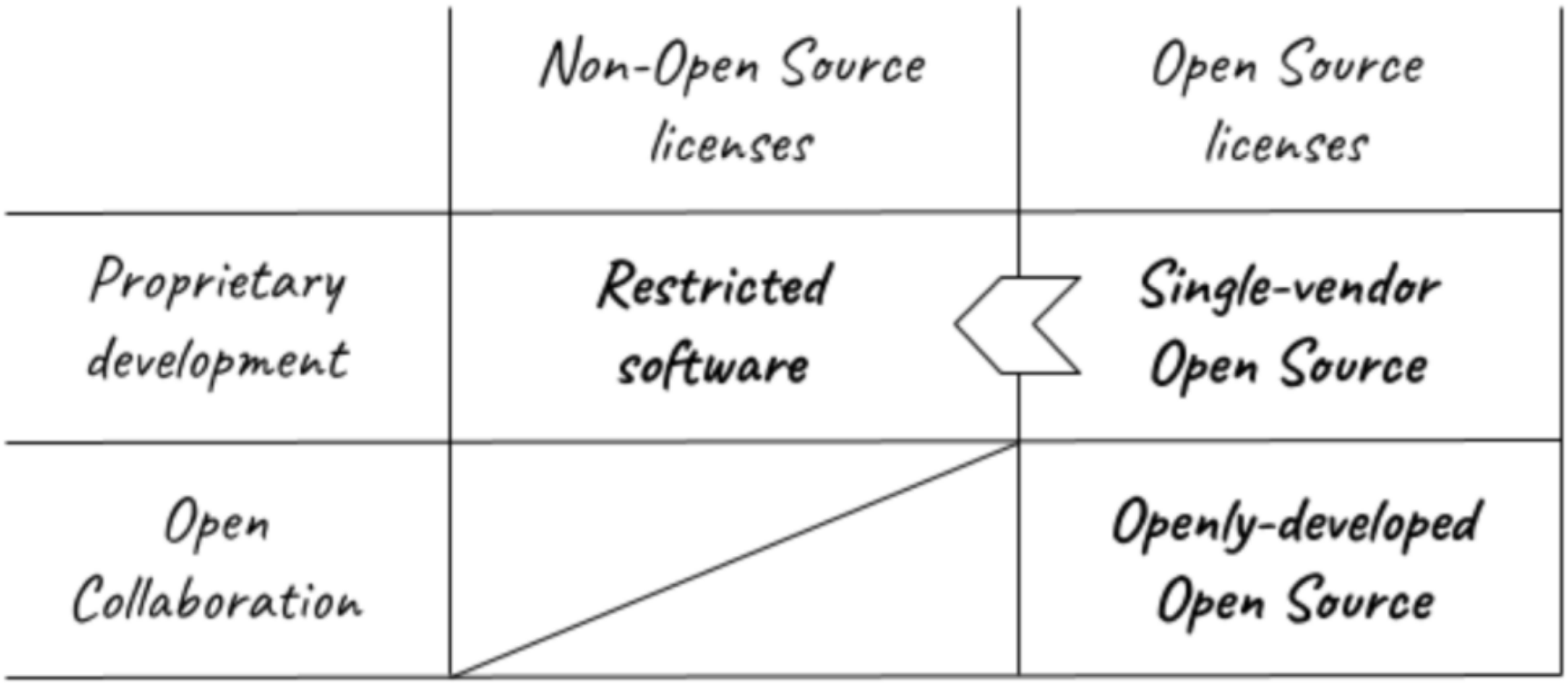


Figure 8: Diagramme issu de l'article "Why single vendor is the new proprietary" par Thierry Carrez (CC BY-SA 4.0)

Section 4

Communication

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

- Essayez de trouver la réponse par vous-même *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/

Subsection 1

Forges et gestionnaires de version

SourceForge

Figure 9: 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

Figure 10: 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

Figure 11: 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).

Subsection 2

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, 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…),
   - séquence d'étapes précise / fichier permettant de **reproduire le problème** (le plus court possible).

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…),
   - séquence d'étapes précise / fichier permettant de **reproduire le problème** (le plus court possible).
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

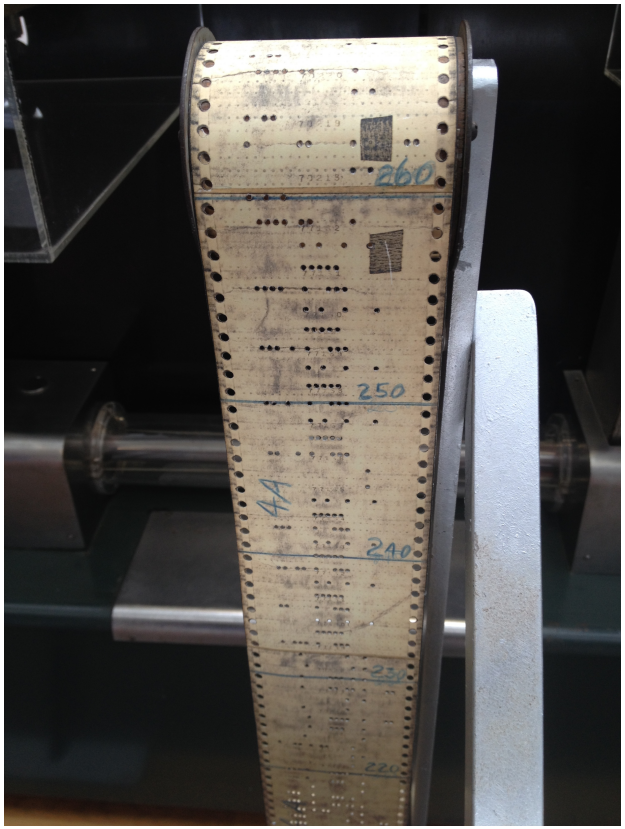


Figure 12: Le mot patch trouverait son origine dans la manière dont on corrigeait les bugs dans les programmes sur ruban perforé.

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.

Des informations supplémentaires sur
plusieurs lignes."
$ 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**.

Section 5

Modèles de gouvernance, communautés

Subsection 1

Communautés

Communautés

En général, un projet se structure autour d'une *communauté*.

Communautés

En général, un projet se structure autour d'une *communauté*.

- **Utilisateurs/ices** (et **clients**)
- **Contributeurs/ices**

Communautés

En général, un projet se structure autour d'une *communauté*.

- **Utilisateurs/ices** (et **clients**)
- **Contributeurs/ices**

On parle d'**organisations** pour les entités capables de gérer les aspects légaux : dons, contrats, etc.

- **Associations**, **entreprises**, **fondations**, **institutions**, etc.

Communautés

En général, un projet se structure autour d'une *communauté*.

- **Utilisateurs/ices** (et **clients**)
- **Contributeurs/ices**

On parle d'**organisations** pour les entités capables de gérer les aspects légaux : dons, contrats, etc.

- **Associations**, **entreprises**, **fondations**, **institutions**, etc.

On observe une grande variété de situations : plusieurs projets peuvent partager une communauté, etc.

Interactions

Les **individus** qui forment une communauté interagissent selon des normes sociales implicites ou explicites. Lorsqu'elles sont explicites, on parle souvent de **code of conduct** (code de bonne conduite).

Exemples : Ubuntu, Django

Ces normes définissent aussi les interactions pouvant arriver entre différentes catégories de personnes : salariés, bénévoles, etc.

Types de contributions

Lister les manières de *contribuer* à un projet

Identification des acteurs dans un projet

Collaboration $\neq$ ***Égalité***

Qui contrôle le projet ? Qui prend les décisions ? Quels sont les rôles de chacun ? Quelles sont les organisations impliquées, et quels sont ses intérêts ?

Le droit au fork

Un *fork* est une copie du code qui diverge du projet original.

Un fork peut avoir plusieurs usages :

- **Contribuer** des changements dans le cadre d'un développement basé sur les pull requests (*development fork*).
- **Maintenir** un projet abandonné par son auteur initial.
- Créer un **projet concurrent** dirigé par une équipe indépendante (*hard fork*).

Les licences libres **garantissent le droit** de distribuer des copies modifiées, donc de créer un fork.

Le droit au fork

Ce droit a un impact sur :

- La **durabilité** :

Le droit au fork

Ce droit a un impact sur :

- La **durabilité** :

  Même si une entreprise décide unilatéralement de cesser de développer un logiciel libre, un fork peut permettre de **préserver** ce logiciel et de **continuer son développement**.

Le droit au fork

Ce droit a un impact sur :

- La **durabilité** :

  Même si une entreprise décide unilatéralement de cesser de développer un logiciel libre, un fork peut permettre de **préserver** ce logiciel et de **continuer son développement**.

- La **confiance** :

Le droit au fork

Ce droit a un impact sur :

- La **durabilité** :

  Même si une entreprise décide unilatéralement de cesser de développer un logiciel libre, un fork peut permettre de **préserver** ce logiciel et de **continuer son développement**.

- La **confiance** :

  Les utilisateurs et les contributeurs peuvent être rassurés que si le projet s'arrête ou prend une mauvaise direction, il sera toujours **possible d'y remédier**.

Le droit au fork

Ce droit a un impact sur :

- La **durabilité** :

  Même si une entreprise décide unilatéralement de cesser de développer un logiciel libre, un fork peut permettre de **préserver** ce logiciel et de **continuer son développement**.

- La **confiance** :

  Les utilisateurs et les contributeurs peuvent être rassurés que si le projet s'arrête ou prend une mauvaise direction, il sera toujours **possible d'y remédier**.

- La **gouvernance** :

Le droit au fork

Ce droit a un impact sur :

- La **durabilité** :

  Même si une entreprise décide unilatéralement de cesser de développer un logiciel libre, un fork peut permettre de **préserver** ce logiciel et de **continuer son développement**.

- La **confiance** :

  Les utilisateurs et les contributeurs peuvent être rassurés que si le projet s'arrête ou prend une mauvaise direction, il sera toujours **possible d'y remédier**.

- La **gouvernance** :

  Même lorsqu'un projet libre est officiellement dirigé par un "dictateur", sa gouvernance est toujours **partiellement démocratique / méritocratique**.

Apache : un fork communautaire

Figure 13: By The Apache Software Foundation and Vulphere, Apache License 2.0

- NCSA HTTPd était l'un des tout premier serveur web et le plus utilisé dans le monde de 1993 à 1995. (Développé à l'Université d'Urbana-Champaign par Robert McCool.)
- Le projet ayant cessé d'être maintenu, ses utilisateurs décident d'en produire un **fork** en aggrégeant des patchs qui circulaient.
- Le nom est un jeu de mot accidentel avec "a patchy server".
- **Serveur web le plus utilisé** dans le monde de 1995 à 2016.
- Donne lieu à la création de l'**Apache Foundation** et de la **licence Apache 2.0**.

Exemple de hard fork : ECGS

- GCC était maintenu par la FSF, de manière très **conservatrice** sur l'ajout de nouvelles fonctionnalités.
- EGCS (Experimental/Enhanced GNU Compiler System) démarra comme un fork rassemblant divers changements qui avaient été proposés par des contributeurs et continua sur un mode de **développement plus ouvert** (mais tout en restant synchronisé avec les modifications faites dans GCC).
- Des distributions Linux commencèrent à adopter EGCS à la place de GCC comme compilateur C.
- La FSF reconnut la validité du modèle de développement d'EGCS et donna le contrôle du projet GCC aux mainteneurs d'EGCS, **mettant fin à la divergence**.

Subsection 2

Modèles de gouvernance

Modèle Linux : dictateur bienveillant

- Traditionnellement, de nombreux projets open source fonctionnent avec un "**dictateur bienveillant**". Dans le projet Python, le titre était "BDFL" qui veut dire "Benevolent Dictator For Life".
- Le dictateur tire sa légitimité de ses compétences techniques et humaines et de son **implication de long terme** dans le projet. C'est très souvent l'**auteur initial** du projet.
- Le dictateur est **celui/celle qui décide** quelles modifications sont acceptées dans le projet. Mais la plupart du temps, ce pouvoir est **délégué** :
  - Par exemple, dans le modèle de gouvernance de Linux, Linus Torvalds est le dictateur et ses "lieutenants" sont les mainteneurs de divers composants du noyau.

Modèle Apache : méritocratie hiérarchisée

Le modèle de gouvernance de la Fondation Apache définit une hiérarchie de rôles au sein d'un projet :

- **Utilisateurs** : participent aux discussions/à rapporter des bugs.
- **Contributeurs** : participent à l'évolution du projet par des contributions de code ou de documentation.
- **Committers** : ont un accès en écriture au projet.
- Membres du **PMC** (Project Management Committee) : dirigent le projet et décident de son évolution.

Le PMC peut rejeter une contribution d'un committer, mais ne peut forcer personne à faire une contribution, c'est pourquoi ce modèle (comme dans tout projet open source) est aussi une "do-ocracy" (**le pouvoir appartient à ceux qui agissent**—*do*).

Cooptation

- Acquérir des **droits en écriture** (commit) sur le projet est un signe de confiance de la part des autres mainteneurs.
- En général, on ne les accorde qu'à des **contributeurs réguliers**.
- Les mainteneurs (ou le dictateur, ou le PMC) **décident quand proposer** à des contributeurs d'acquérir ces droits. Cela peut faire suite à une discussion dont il est naturel qu'elle ait lieu **en privé**.
- Il peut y avoir une distinction entre des **droits limités** (maintenance d'un composant) et des droits globaux (sur tout le projet).
- Le PMC (ou équivalent) peut être renouvelé également par cooptation ou être élu (par un électorat composé des committers ou plus vaste).

Exemple : projet Python

- Guido van Rossum (créateur du langage) était le BDFL jusqu'à ce qu'il **démissionne** en 2018.
- La gouvernance actuelle (inspirée de Django) est composée :
  - D'une **équipe cœur** (106 membres actifs).
  - D'un **comité de direction** (5 membres).
- Les nouveaux membres de l'équipe cœur doivent être validés par un vote de 2/3 des membres actifs.
- Le comité de direction est **élu par l'équipe cœur** et renouvelé régulièrement ($\sim$ tous les ans). Il a un fort **pouvoir décisionnaire**, à n'utiliser qu'en dernier recours.
- Pas plus de deux membres du comité de direction ne peuvent avoir le même employeur.

Le consensus paresseux

- Quel que soit le modèle exact de gouvernance d'un projet open source, il est standard que la plupart des décisions soient prises par **consensus paresseux** :
  - Les oppositions doivent être **explicites**.
  - Les silences sont considérés comme des approbations.
  - Les oppositions sont prises au sérieux et quasiment équivalentes à des **vetos** (selon par qui elles sont émises).
- Selon l'importance de la décision : **processus + ou − formel**.
  - Le − formel : agir (pousser un changement) et annuler l'action en cas de protestation (*revert*).
  - Le + formel : déclarer la décision qui va être prise si personne ne s'y oppose pendant une période définie (de plusieurs jours), *"Final Comment Period"*.

Les PEP / RFC

Certains projets (dont des langages de programmation comme Python, Go et Rust) ont adopté un modèle de décision pour les **modifications importantes** au code (ou à la gouvernance) reposant sur des **discussions préalables sur des documents**, PEP (Python Enhancement Proposals) ou RFC (Request For Comments) :

- **Un projet de modification est présenté** avec des motivations, des explications sur les choix effectués, les alternatives et les conséquences qui ont été considérées.
- Une **discussion** a lieu sur les forums / mailing lists qui peut conduire à la **mise à jour du document**.
- Un processus de décision (ex. recherche d'un consensus paresseux, validation par l'équipe cœur, etc.) permet d'**entériner la proposition**, qui pourra ensuite être implémentée par des volontaires.

Méritocratie/Do-ocracy : un système parfait ?

Quels sont les problèmes posés par les systèmes de do-ocracy ? de BDFL ?

Section 6

Rôles et parcours de contributeurs

Pourquoi contribuer à des logiciels libres ?

Pourquoi contribuer à des logiciels libres ?

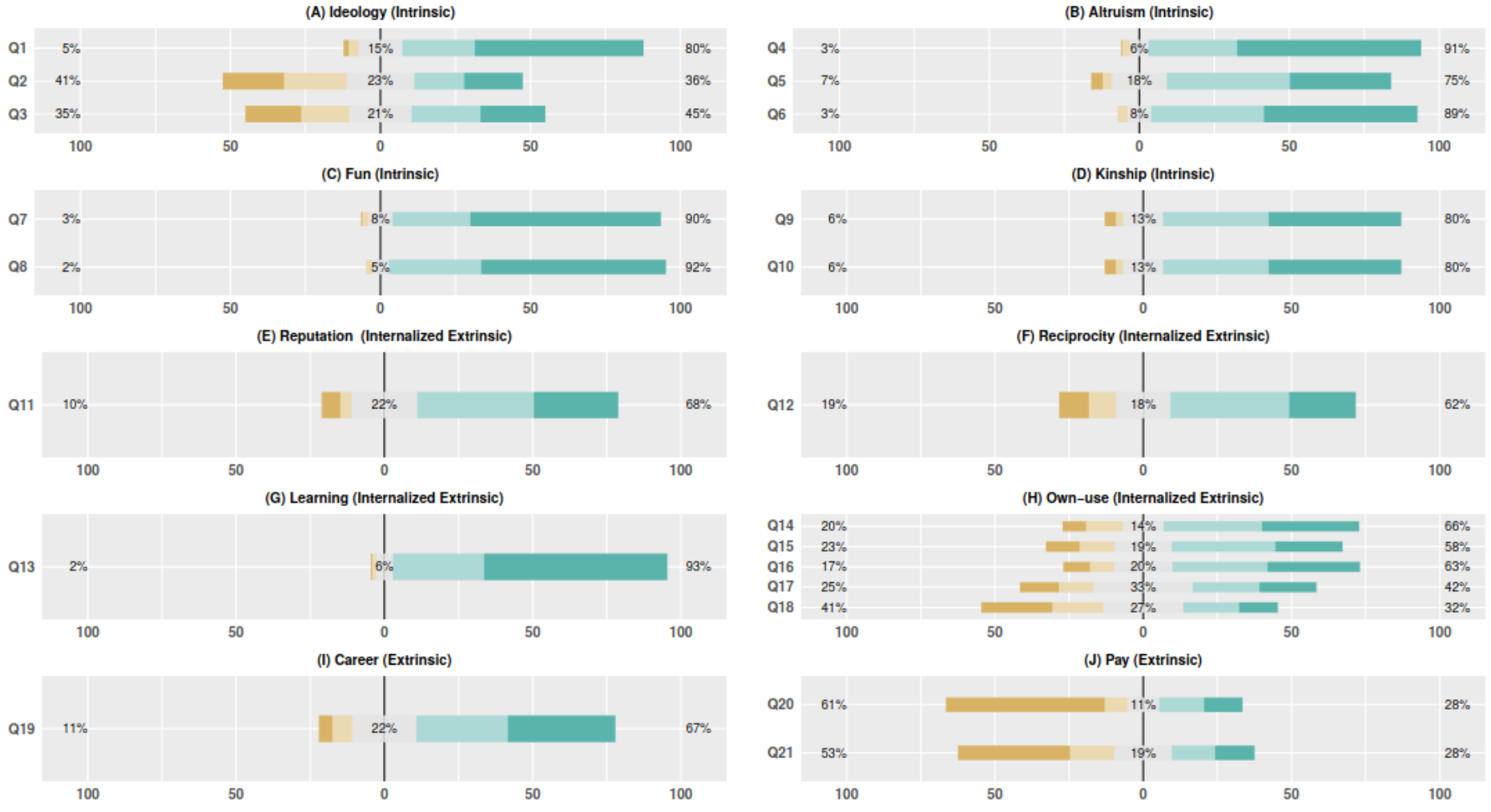


Figure 14: Motivations des contributeurs open source (le bleu/vert représente l'accord, le jaune le désaccord). Extrait de Gerosa et al. (2021).

Contributeurs occasionnels

Beaucoup de contributions se font par des contributeurs **occasionnels** (drive-by)

- Utiliser un outil
- Constater un bug
- Récupérer le source
- Corriger le bug
- Faire une MR/PR
- La faire accepter

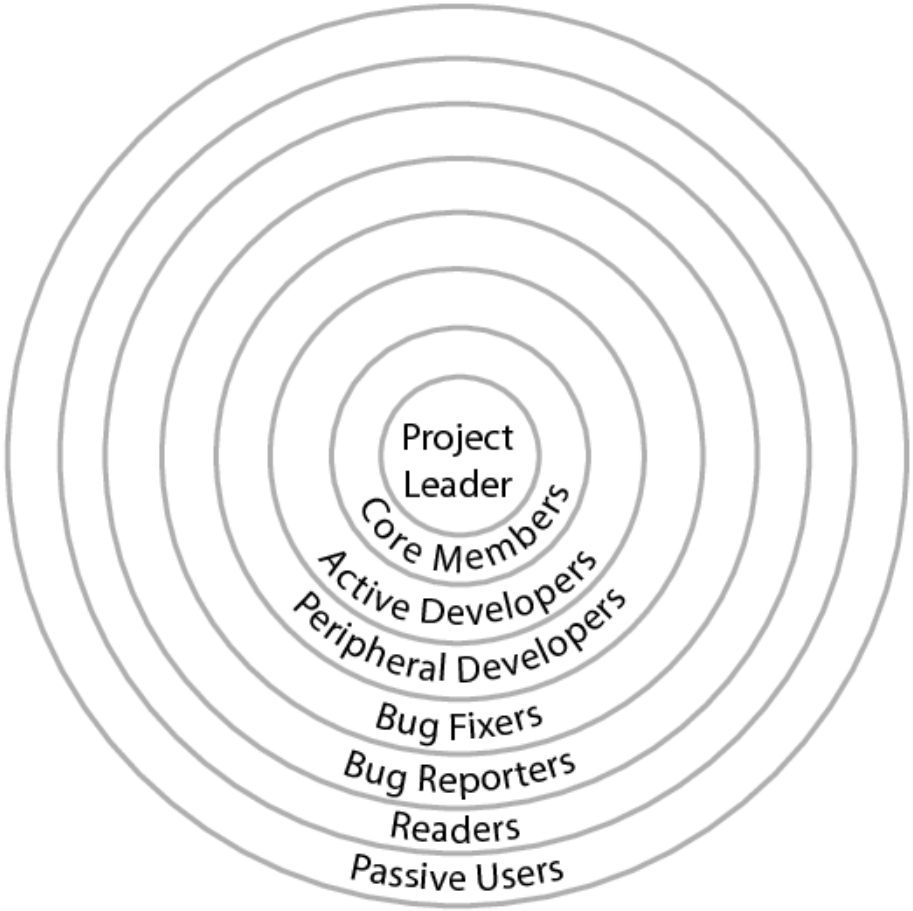
Contributeurs réguliers et mainteneurs

Les contributeurs *réguliers* sur un projet ont en général plus d'impact long-terme sur

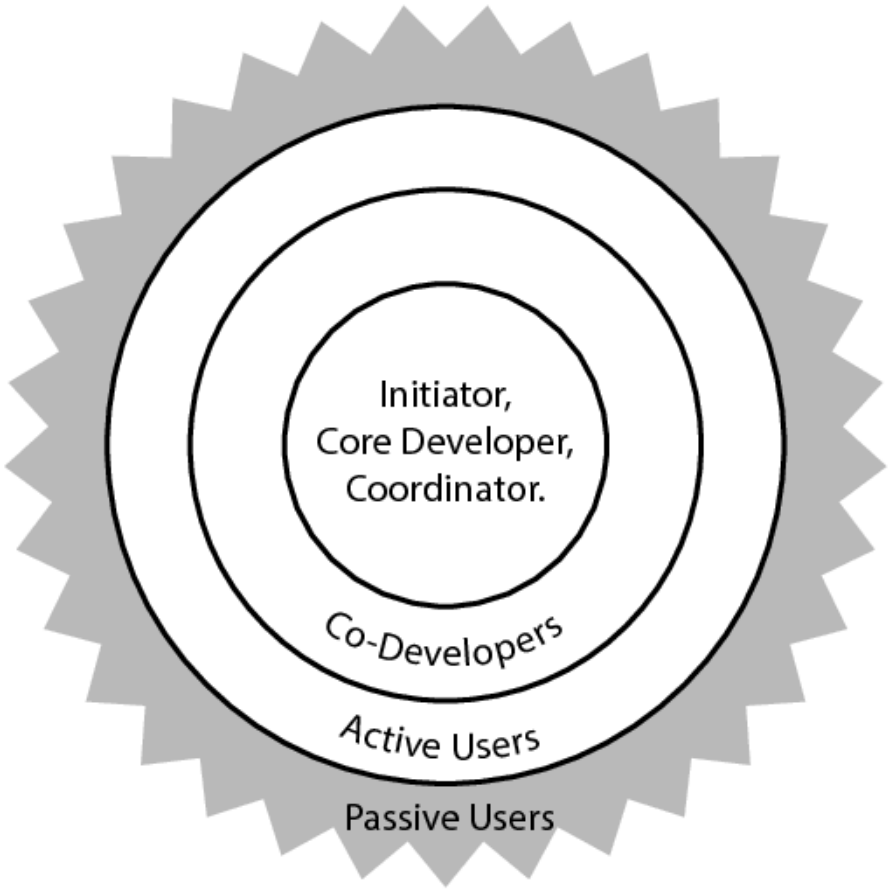
- L'évolution future du projet (*roadmap*)
- La maintenance
- La gestion de la communauté et l'accueil des contributeurs

Un *mainteneur* est un contributeur régulier qui partage la charge de la gestion du projet et prend des décisions.

Structure en oignon



Seven layer community Structure.
From Nakakoji et al (2002).



Four layer team structure.
From Crowston and Howison (2005)

Section 7

Modèles économiques

Subsection 1

L'open source à la conquête des entreprises

The Cathedral and the Bazaar

Essai d'Eric Raymond (1997) comparant deux modèles de développement :

- La **cathédrale** : modèle de développement traditionnel, "vertical".

  Exemples : Emacs, gcc (à cette époque)...

- Le **bazar** : modèle de développement flexible, "horizontal" (aujourd'hui on dirait "agile").

  Un principe : "Release early, release often".

  Et une loi : "Given enough eyeballs, all bugs are shallow".

  Exemples : Linux, fetchmail (expérience d'application du modèle).

Mozilla

Figure 15: By Mozilla Corporation, MPL 2.0

- Netscape : navigateur dominant entre 1995 et 1997, quand Internet Explorer écrase le marché en étant **distribué avec Windows**.
- En 1999, l'entreprise **Netscape libère le code de son navigateur** qui devient Mozilla (puis Mozilla Firefox).
- Firefox parvient à être un **concurrent sérieux d'Internet Explorer** jusqu'à ce que Google Chrome arrive et écrase tout à son tour.
- Création de la **Mozilla Foundation** en 2003.
- À l'origine de la **licence MPL 2.0**.

StarOffice, OpenOffice, LibreOffice

Figure 16: By Christoph Noack, CC BY-SA 3.0

- Star Division, une entreprise allemande publiant la suite StarOffice, est rachetée en 1999 par Sun Microsystems.
- En 2000, **Sun Microsystems libère le code** de la suite qui devient OpenOffice.
- Création en 2005 du format standard OpenDocument.
- **Oracle rachète Sun Microsystems** en 2009.
- The Document Foundation est créée en 2010 pour créer un **fork communautaire** : LibreOffice.

Android

Figure 17: By Google, CC BY-SA 3.0

- Système d'exploitation open source **basé sur Linux**, développé par Google et publié à partir de 2007.
- Permet de **concurrencer Apple** et l'iPhone (plus de 70% du marché).
- N'est pas développé de manière collaborative ou ouverte, mais la licence **autorise les fabricants** de smartphones à en produire leur **version dérivée**.
- En pratique, la plupart des smartphones sous Android sont **loin d'être libres** car ils incluent aussi les Google Mobile Services (Google Play) qui sont propriétaires.

Microsoft et l'open source

- Open Letter to Hobbyists (1976) : Bill Gates se plaint du "**vol**" de ses logiciels.
- Steve Balmer compare Linux à "un **cancer** qui s'attache à tout ce qu'il touche" (2001).
- Microsoft commence à contribuer à Linux en 2009, publie des langages de programmation open source (F# en 2005, **TypeScript** en 2012).
- Satya Nadella écrit "Microsoft <3 Linux" en 2014 (Microsoft fait tourner Linux dans Azure depuis 2012), Microsoft devient l'un des plus gros **sponsor de la Linux Foundation** en 2018.
- **VS Code** est publié en open source en 2015.
- Le **Windows Subsystem for Linux** est introduit en 2016.
- Microsoft rachète **GitHub** en 2018.

Le logiciel libre a-t-il gagné ?

- On pourrait croire que oui si on regarde les **outils utilisés par les développeurs** sur leurs ordinateurs (éditeurs, gestionnaires de version, compilateurs, bibliothèques).
- Mais si on regarde en termes de **libertés des utilisateurs finaux**, on en est loin !
- La plupart des applications web et mobiles **dépendent de logiciels libres** mais ne donnent pas de libertés à l'utilisateur.
- GitHub n'est pas libre ! (Même si des alternatives libres, comme GitLab, Gitea, Forgejo, etc. existent.)
- Et Microsoft Word continue aussi à dominer le marché face à LibreOffice.

Subsection 2

Modèles économiques

Le modèle classique

- Dans le monde des logiciels propriétaires:
  Revenus basés sur la **vente de licences** d'utilisation.
- Production de logiciels: coût élevé de développement, coût de la copie très bas (quasiment nul).
- C'est un modèle de *rente* qui peut être extrêmement profitable (dépend davantage du nombre de clients que des efforts fournis par le vendeur).
- Modèle similaire pour la musique, les films, etc.

L'exclusion d'usage commercial

- Logiciel **gratuit pour les usages non commerciaux** (ou pour les très petites entreprises / indépendants) mais payant pour les usages commerciaux (ou au-delà d'une certaine échelle).
- Cela permet d'avoir une plus large communauté d'utilisateurs (voire de contributeurs) sans renoncer à l'économie de rente.
- Même lorsque le code est public, ce n'est **pas du logiciel libre**, et beaucoup d'utilisateurs ou contributeurs potentiels en auront conscience. Par exemple :
  - **Pas de garantie sur l'avenir du logiciel** en cas de faillite / renoncement / rachat de son producteur.
- Exemples actuels : SublimeText (éditeur), Obsidian (notes).

Modèles économiques du logiciel libre

1. Les modèles basés sur le *développement* payant et non la *vente*.
2. Les modèles basés sur la vente de versions propriétaires étendues par rapport à une version libre et gratuite (*open core*) ou bien de licences spécifiques.
3. Les modèles qui ne sont pas basés sur la vente des logiciels mais sur des services annexes : maintenance, hébergement, etc.

Chaque acteur peut combiner plusieurs modèles à sa guise. Dans la suite, on se focalise sur des *éléments* constitutifs de modèles économiques.

Développement payant

- Un client a besoin d'un certain produit logiciel.
- Il peut embaucher un développeur, ou une entreprise (éditeur de logiciel) qui développe le logiciel.
- Le client (qui est *a priori* propriétaire du logiciel qu'il a fait développer à ses frais) peut choisir de publier le produit sous licence libre pour des raisons diverses.
- Modèle fréquent lors de la commande de logiciels par des institutions publiques (notamment aux US).

Vente d'extensions propriétaires (modèle *open core*)

- Deux versions :
  - Une version libre.
  - Une version propriétaire **étendue avec plus de fonctionnalités** (qui peuvent finir par être ajoutées à la version libre).
- Les entreprises sont encouragées à acheter la version étendue :
  - Choix d'un vocabulaire (trompeur) :
    "Community Edition" / "Enterprise Edition"
  - **Vente combinée** de services (déploiement/maintenance) avec les extensions.
- Avantages :
  - Si la version libre n'est pas sous licence copyleft, alors le recours à un CLA / CTA n'est pas nécessaire.
  - Les distributions Linux peuvent inclure la version libre, qui sera néanmoins à jour.
- Exemple : Modèle passé de Netscape (licence MPL), actuel de GitLab (licence MIT).

Vente d'exceptions

- Publication sous licence libre *copyleft* (par exemple GPL).
- Vente de licences qui permettent l'utilisation du logiciel dans des produits qui ne seront pas publiés sous GPL.
- C'est un modèle que la FSF trouve acceptable, contrairement au précédent (mais qui requiert un CLA ou CTA) : https://www.gnu.org/philosophy/selling-exceptions.html
- Exemple : 

un des systèmes de bases de données les plus importants (entreprise rachetée par Sun pour un milliard de dollars, et ensuite par Oracle).

Vente de maintenance / support : exemple de RedHat

- Fondé en 1994.
- Chiffre d'affaires : 1 milliard de $ en 2012.
- Résultat net : 199 millions de $ en 2012.
- Nombre d'employés : 6 100 en 2014.
- Vente de support : entre 350$ et 2 500$ par an et serveur.
- Basée sur une distribution gratuite et libre : *Fedora*
- 82% des revenus (2012) viennent de la vente de contrat de support, le reste de la formation et du conseil.
- 18% des coûts (2012) sont pour le développement Open Source (les résultats reviennent donc à la communauté).

Hébergement (modèle SaaS) : exemple de Zulip

- SaaS = Software as a Service
- Zulip est un logiciel libre de chat / alternative à Slack très populaire depuis quelques années.
- L'entreprise Zulip propose d'héberger des serveurs pré-configurés et automatiquement mis à jour :
  - Gratuitement pour les projets open source, les équipes de recherche...
  - De manière payante pour les entreprises.

Financement par les dons

Désormais communément admis que les projets open source ont besoin d'être soutenus financièrement.

Quelques succès célèbres comme celui d'Evan You, le créateur de Vue.js.

Mais, d'après Overney et al. (2020) :

- Beaucoup de projets demandent des dons pour supporter les activités de développement.
- Mais **rares sont ceux qui reçoivent suffisamment** pour financer un développeur à plein temps. Par exemple sur npm :
  - Seuls 2 paquets sur 1000 réclament des dons.
  - Seuls 5% de ceux qui en réclament reçoivent $>$ 1000\$/mois.
- Beaucoup de projets (sur Open Collective) ne dépensent pas l'argent collecté (thésaurisation) ou le dépensent pour d'autres utilisations (exemple : frais d'hébergement).

Financement ponctuels par des bourses / crowdfunding ...

- Bourses de fondations :

  Exemple : NLNet

  - une fondation soutenant l'internet ouvert,
  - aujourd'hui distribue à + de 90% de l'argent de l'UE (**Next Generation Internet** initiative),
  - a financé des dizaines de projets open source (sur des missions ponctuelles et bien définies)

- **Financement participatif** (crowdfunding) : une campagne de dons ponctuelle pour mobiliser la communauté des utilisateurs afin de financer une amélioration spécifiques.

- Autres : missions diverses, bounties, etc.

Exercice

Quel est le modèle de financement de Firefox ? GIMP ? Krita ? Blender ? Gitlab ? Linux ?