git-bug: un bug tracker distribué intégré dans git

Posté par MichaelMure (page perso) le 06/12/18 à 11:51. Édité par ZeroHeure et palm123. Modéré par Pierre Jarillon.
Tags : git, bugtracker

À l'occasion de la sortie de la version 0.4 de **git-bug**, je me suis dit que je pourrai te raconter un peu de quoi il retourne. **git-bug** est un gestionnaire de bugs distribué intégré dans git.

git-bug est un **bug tracker**[W] distribué intégré dans git, c'est-à-dire que l'ensemble des données des bugs est stocké sous forme d'objet **git**, sans polluer les branches et fichiers normaux. Mais pour quoi faire ? Et bien par rapport à un bug tracker classique, il y a plusieurs avantages :

- pas de dépendance à un service en particulier (par exemple si ton bugtracker préféré se fait racheter par *tousse* Microsoft *tousse*) ;
- pas besoin d'infrastructure particulière : si tu utilises déjà un remote **git** pour collaborer, tu as un bugtracker ;
- tu peux consulter et éditer les bugs hors-ligne, avec l'interface que tu préfères, sans latence.

Mais alors, comment on fait un bug tracker distribué ? Le plus gros problème c'est la gestion des conflits. Comment faire si Alice et Bob modifient le même bug et s'échangent les données ensuite ?

La solution choisie, plutôt que de stocker simplement l'état d'un bug, est de stocker chaque opération d'édition individuellement.

```
      +-------------+    +-----------+    +-------------+   +--------+
* --->| ADD_COMMENT |--->| SET_TITLE |--->| ADD_COMMENT |-->| CREATE |
      +-------------+    +-----------+    +-------------+   +--------+
```

Quand une fusion doit avoir lieu, l'une des deux versions voit ses nouvelles opérations placées à la fin de la chaine, à la manière d'un rebase. On a donc la garantie d'avoir un état final cohérent puisque c'est **git-bug** qui dérive l'état final de la chaîne d'opérations. Mais que se passe-t-il si deux personnes suppriment la même étiquette ? Elle sera simplement supprimée une fois et l'action des deux personnes sera visible dans l'historique.

Voilà à quoi pourrait ressembler une opération :

```
{
  "type": "SET_TITLE",
  "author": {
    "name": "René Descartes",
    "email": "rene.descartes@example.com"
  },
  "timestamp": 1533640589,
  "title": "This title is better"
}
```

Ces opérations sont ensuite sérialisées et stockées dans des **blob** git et reliées dans une chaîne de **commits**. Chaque bug a sa propre chaîne de commits, accessible via une **référence** git. À chaque fois que le bug est modifié, une nouvelle série d'opérations est ajoutée à la chaîne et la référence est mise à jour.

```
 refs/bugs/<bug-id>
       |
       |
       v
 +-----------+    +-----------+             "ops"    +-----------+
 |  Commit   |--->|   Tree    |---------+----------->|   Blob    | (OperationPack)
 +-----------+    +-----------+         |            +-----------+
       |                                |
       |                                |
       v                                |   "root"   +-----------+
 +-----------+    +-----------+         +----------->|   Blob    | (OperationPack)
 |  Commit   |--->|   Tree    |-- ...   |            +-----------+
 +-----------+    +-----------+         |
       |                                |
       |                                |   "media"  +-----------+        +-----------+
       v                                +----------->|   Tree    |---+--->|   Blob    | bug.jpg
 +-----------+    +-----------+                      +-----------+   |    +-----------+
 |  Commit   |--->|   Tree    |-- ...                                |
 +-----------+    +-----------+                                      |    +-----------+
                                                                     +--->|   Blob    | demo.mp4
                                                                          +-----------+
```

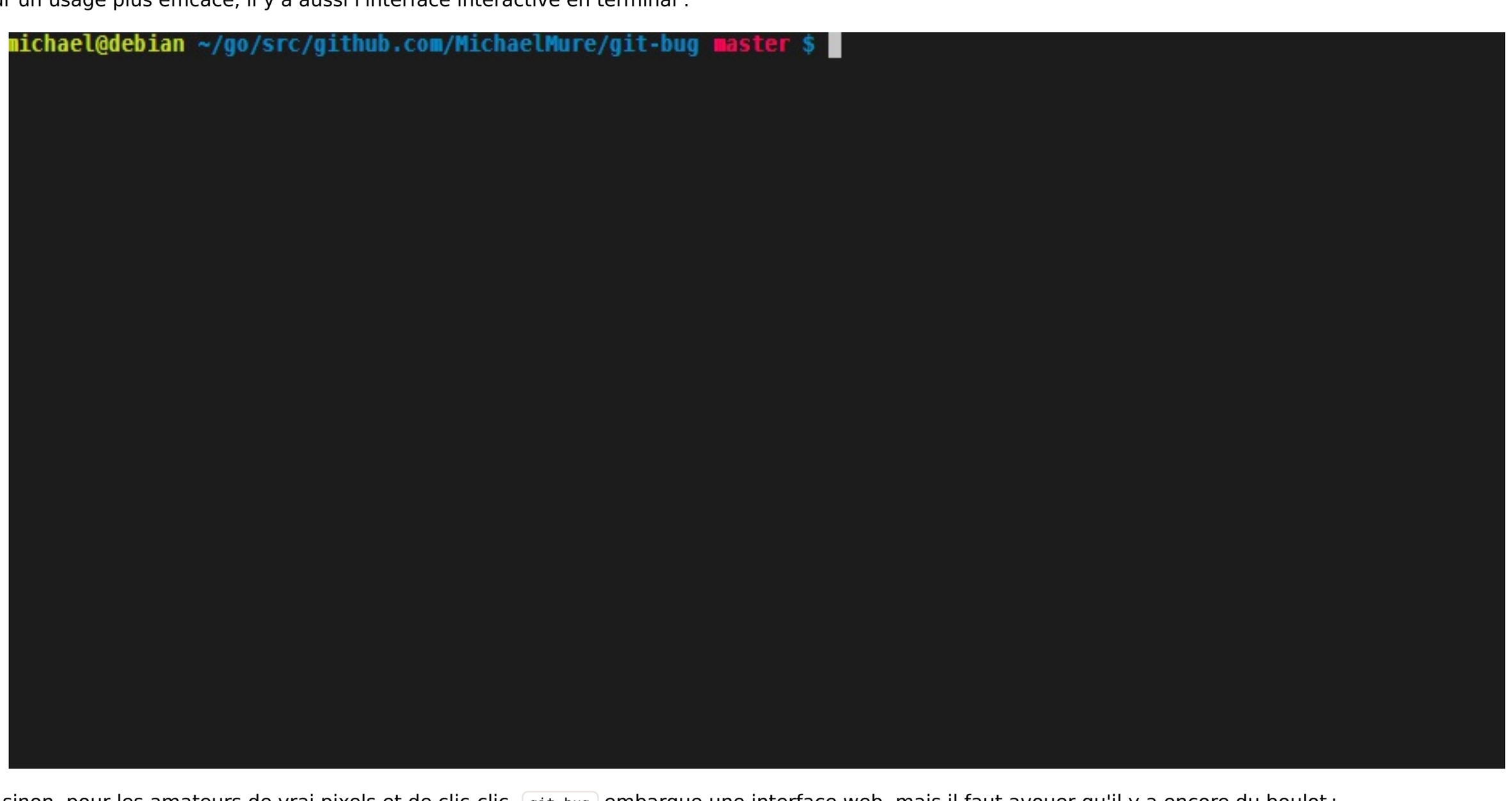
Comme tout est stocké sous forme d'objet git, on peut pousser tout ça vers un **remote** et collaborer, comme si on s'échangeait du code source.

Une autre propriété intéressante de la méthode, c'est qu'on peut stocker et lier les médias (typiquement, une capture d'écran) à un bug et les diffuser de la même façon.

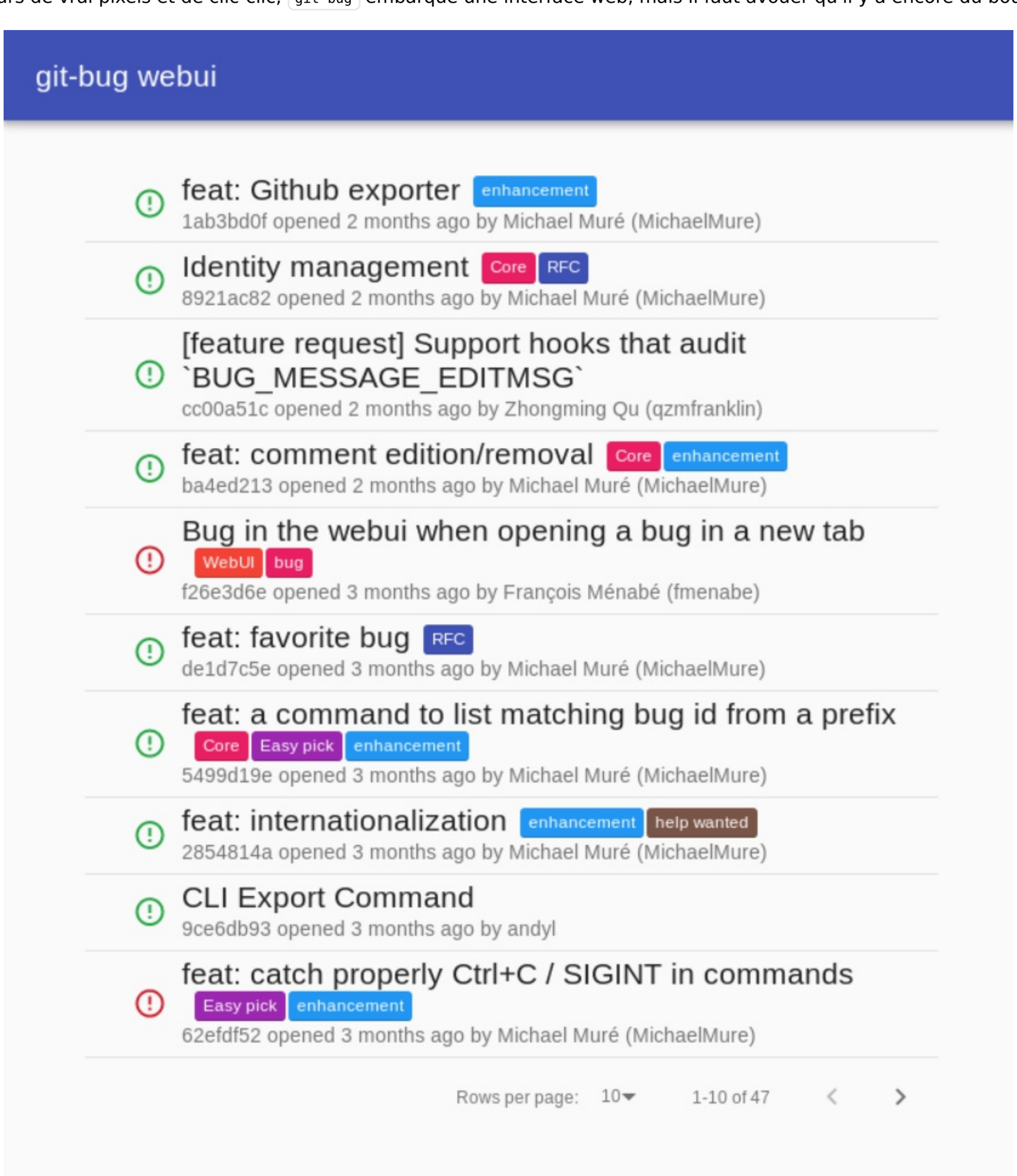
Alors, à quoi ça ressemble tout ça ? Et bien ça dépend des goûts. Si tu préfères la ligne de commande, le scripting ou intégrer ça dans ton Vim, c'est possible. Voilà un aperçu:

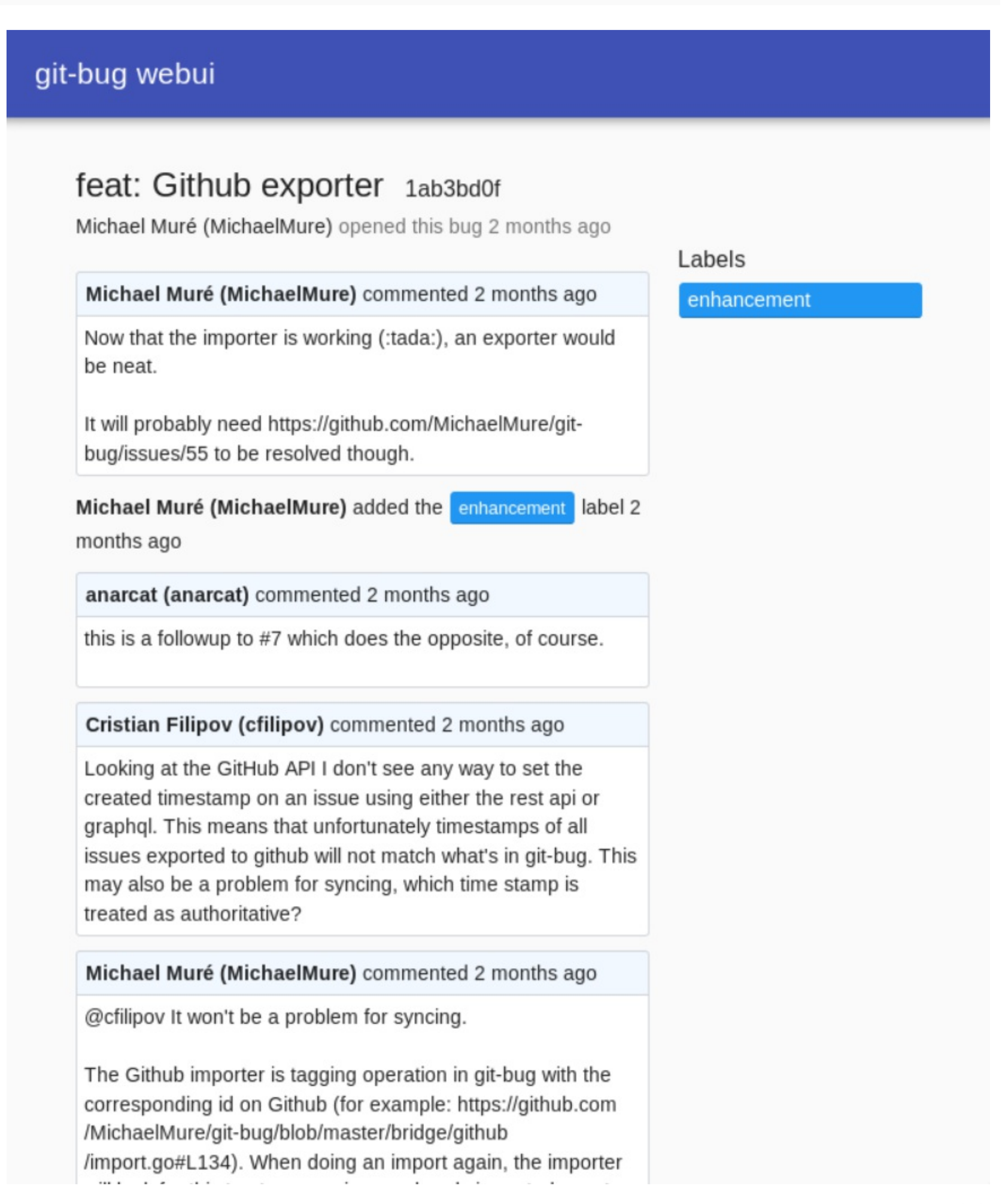
```
# Ajouter un nouveau bug
git bug add

# Liste les bugs ouverts, trié par dernière édition
git bug ls "status:open sort:edit"

# Affiche l'état du bug f3781aee
git bug show f3

# Selectionne le bug f3781aee parce que je sais bien que tu es un peu flemmard
git bug select f3

# Ajoute un nouveau commentaire
git bug comment add

# Pousse les modifications vers le remote par défaut
git bug push
```

Pour un usage plus efficace, il y a aussi l'interface interactive en terminal :



Ou sinon, pour les amateurs de vrai pixels et de clic-clic, `git-bug` embarque une interface web, mais il faut avouer qu'il y a encore du boulot :





Techniquement, le binaire de **git-bug** lance un serveur web local qui sert les fichiers d'une application REACT (JavaScript) qui ont été compilés et inclus dans ce même binaire. L'application web s'interface avec le binaire grâce à une **API GraphQL**. Le déploiement reste donc extrêmement simple avec un unique binaire à copier, sans configuration.

La nouvelle version 0.4 apporte une fonctionalité qui a été beaucoup demandée, un importeur incrémental pour Github ! Après avoir utilisé `git bug bridge configure` pour saisir ses identifiants, un token d'API est généré automatiquement et **git-bug** devient capable d'aspirer les issues en local dans son modèle de données.

Et le futur ? Une de mes priorités est d'implémenter un exporteur vers Github. Quand ça sera fait, il sera possible d'utiliser **git-bug** comme un Github déporté, qui fonctionne en local et offline.

Une autre idée qui me trotte dans la tête est d'avoir une véritable gestion d'identité en embarquant les données publiques dans des objets git, clé de chiffrement incluse. De cette façon, il devient possible de les distribuer facilement, avec les données des bugs, de signer ses éditions, les vérifier, et même pourquoi pas ? de vérifier les commits git normaux sans avoir à chercher et importer les clés GPG.

Mais tout ça, c'est quand même beaucoup pour, en gros, un seul bonhomme. Mais peut-être, mon cher journal, que je trouverai des âmes vaillantes pour m'accompagner dans cette aventure…

Aller plus loin

- **Journal à l'origine de la dépêche** (60 clics)
- **Le projet git-bug** (221 clics)
- **Téléchargement et versions** (40 clics)

La taille ça compte...

Posté par Flyounet (page perso) le 06/12/18 à 12:33. Évalué à 5 (+4/-0).

J'aime bien l'idée.

Quelle est l'espace utilisé par une centaine de bugs avec quelques images (disons une vingtaine) ?

Admettons que j'ai deux repositories dans lesquels je push. Des utilisateurs différents sur chaque remote.
Les bugs vont-ils être mergés ou font-ils parti de "namespace" différents ?

Re: La taille ça compte...

Posté par MichaelMure (page perso) le 06/12/18 à 13:08. Évalué à 8 (+7/-0). Dernière modification le 06/12/18 à 13:08.

> *Quelle est l'espace utilisé par une centaine de bugs avec quelques images (disons une vingtaine) ?*

J'ai fais un test avec 10.000 bugs, 10 commentaires par bug (du vrai texte représentatif d'une discussion) et le résultat est de 21Mo après un `git gc` qui compresse les données. Bien sûr, si tu ajoute des images, ça fera gonfler la facture, de la même façon que dans un commit normal.

Cela dit, chaque bug est indépendant, donc il est possible de récupérer qu'une copie partielle d'un dépôt, comme par exemple ignorer des bugs marqué comme "archivé", récupérer uniquement les bugs récents ou ceux où on participe à la discussion.

> *Admettons que j'ai deux repositories dans lesquels je push. Des utilisateurs différents sur chaque remote.*
> *Les bugs vont-ils être mergés ou font-ils parti de "namespace" différents ?*

Si un bug est créer et poussé vers 2 dépôts différents, il a toujours le même identifiants et pourra être fusionné ultérieurement, même si il y a eu des changements des deux cotés. Bien sûr, même si il y aura toujours un état final "légal", l'historique sera probablement différent ce que tu attend. Dans un système distribué, tu ne peux pas te baser sur des dates/horaires pour déterminer l'ordre des évènements, donc il n'est pas possible d'entrelacer les opérations de manière fiable pendant la fusion. Dans la pratique, tu voudra probablement éviter ce scénario.

Re: La taille ça compte...

Posté par Albert_ le 07/12/18 à 11:42. Évalué à 2 (+0/-0).

Mettre des images dans un repo git c'est … une (tres) mauvaise idee!!!

Pour ca il faut passer par git lfs (qui semble tre devenu la norme pour cela) ou git annexe mais voir point precedent.

Re: La taille ça compte...

Posté par cosmocat le 07/12/18 à 12:35. Évalué à 2 (+0/-0). Dernière modification le 07/12/18 à 12:35.

> *Mettre des images dans un repo git c'est … une (tres) mauvaise idee!!!*
> *Pour ca il faut passer par git lfs*

oui, sauf que là, il semble que ce soit `git-bug` qui ajoute directement les fichiers dans la "base de données" de git (cf `refs/bugs/<bug-id>`).

Donc il faut que ce soit `git-bug` qui gère `git-lfs` pour les images.
Ce qui doit être plus compliqué à faire…

Re: La taille ça compte...

Posté par Albert_ le 07/12/18 à 12:51. Évalué à 2 (+0/-0).

oui je suis d'accord mais cela ne change pas que c'est une extremement mauvaise idee de mettre des fichiers binaires dans un repo git.

Enfin, vous faites ce que vous voulez mais le repo va etre assez rapidement bien pourri et nettoyer ce genre de fichier c'est penible.

Excellente idée

Posté par Jokernathan le 06/12/18 à 13:08. Évalué à 10 (+9/-0).

Je trouve l'idée très bonne et (d'après ce que montre la capture) la réalisation plutôt bien foutue notamment l'interface terminal. Et j'adore le logo !

J'espère que tu vas augmenter rapidement ton nombre d'utilisateurs et de contributeurs pour ajouter de nouvelles fonctionnalités. Je pense notamment que tout ce qui est migration et/ou lien vers des systèmes de suivi existant est ce qui va être recherché en premier pour l'adoption de ce type d'outil.

Petite question au passage : le suivi des bugs de git-bug est-il fait avec git-bug ?

Re: Excellente idée

Posté par MichaelMure (page perso) le 06/12/18 à 13:18. Évalué à 10 (+11/-0).

Le suivi des bugs de git-bug n'est pas fait avec git-bug actuellement, pour plusieurs raisons:

- un bug tracker a besoin d'accepter des éditions d'utilisateurs qui n'ont pas les droits d'écriture sur le dépôt. A terme, l'idée est de faire tourner l'interface web comme un portail qui accepte des comptes utilisateurs externe et stocke les changements dans le dépôt git central d'un projet. Mais l'interface n'est pas encore prête pour ça.
- l'idée de git-bug n'est pas forcement de remplacer les bug tracker existants, mais de rendre l'accès et l'édition offline possible, et de casser la dépendance à un service / une entreprise. Je n'ai personnellement pas de soucis particulier avec Github, mais si demain ça change, mes issues sont déjà accessible en local et je peux les migrer ailleurs.

conflit avec git-extras

Posté par autra le 06/12/18 à 15:04. Évalué à 3 (+3/-0).

Bravo, c'est une super idée !!

Par contre: petit problème de nom qui fait conflit avec `git-bug` ou `git bug` fournit par le paquet `git-extras` sous ubuntu, cf **https://github.com/tj/git-extras**

du coup, même en l'ayant installé, je ne peux pas l'avoir dans le path tant que je ne désinstalle pas ce paquet. Ya un workaround ? Est-ce qu'un renommage est envisageable ?

Re: conflit avec git-extras

Posté par MichaelMure (page perso) le 06/12/18 à 15:48. Évalué à 1 (+0/-0).

C'est vrai, il y a un conflit. Je réfléchis à sortir du système de commande git porcelain et à renommer le binaire en `gb` par exemple. Plus de conflit et des commandes plus courte.

Re: conflit avec git-extras

Posté par windu.2b le 06/12/18 à 16:18. Évalué à 3 (+1/-0).

Et en renommant en `git bt` (pour bugtracker) ?

Re: conflit avec git-extras

Posté par MichaelMure (page perso) le 06/12/18 à 17:01. Évalué à 2 (+1/-0).

Ça m'embête un peu de perdre la corrélation entre le nom du projet et la commande :-|

Re: conflit avec git-extras

Posté par Cyrille Pontvieux (page perso) le 07/12/18 à 08:55. Évalué à 1 (+0/-0).

`gb` ce sera pas top non plus. `g bt` reste court est compréhensible je pense (j'ai un alias de git sur g)

Dommage que ce soit git only

Posté par David Demelier (page perso) le 06/12/18 à 16:53. Évalué à 3 (+2/-0).

J'ai toujours aimé l'idée d'avoir un bug tracker directement dans le dépôt. Il y a eu d'autres tentatives comme ditz ou bugseverywhere.

L'interface avait l'air chouette, mais étant utilisateur et contributeur Mercurial j'aurai bien aimé qu'il soit SCM indépendant. Dommage.

--
l'azerty est ce que subversion est aux SCMs

Re: Dommage que ce soit git only

Posté par MichaelMure (page perso) le 06/12/18 à 17:12. Évalué à 4 (+3/-0). Dernière modification le 06/12/18 à 17:12.

Dans l'absolu, ça serait pas forcement difficile de le porter sur Mercurial. Si c'est possible de faire correspondre les concepts et d'implémenter l'interface qui va bien (**https://github.com/MichaelMure/git-bug/blob/master/repository/repo.go**), ça devrait marcher.

Mais je connais que vaguement Mercurial, aucune idée si c'est jouable …

Pourquoi une nouvelle branche?

Posté par freem le 06/12/18 à 22:11. Évalué à 4 (+2/-0). Dernière modification le 06/12/18 à 22:12.

Ce n'est pas le 1er bugtracker intégré au DVCS sur lequel je tombe, parce que, j'en cherche un depuis longtemps (sans trop me forcer quand même), qui me convienne (mais je suis un grand chieur, donc…).
Je m'aperçois que la plupart se concentrent sur le fait d'isoler la gestion des bugs sur une branche différente, et je me demande pourquoi.

Après tout, un bug peut n'être lié qu'à une branche précise, et tant qu'il n'est pas résolu, il devrait parfois pouvoir bloquer la fusion.
Donc, pourquoi ne pas juste créer une arborescence, avec quelques éventuels scripts (inclus via peu importe quel moyen), pour ça?

C'est justement un des trucs qui font que je trouve fossil pas si intéressant que ça, en plus du fait de nécessiter un outillage binaire spécial rien que pour ça.

Re: Pourquoi une nouvelle branche?

Posté par MichaelMure (page perso) le 07/12/18 à 01:40. Évalué à 10 (+10/-0).

Plusieurs raisons:

1. Faire ça dans une branche, ça implique d'avoir des données des bugs au même endroits que le code. C'est à mon avis une mauvaise idée: ça pollue, c'est facile d'éditer les infos des bugs sans faire exprès, et surtout ça veut dire que potentiellement, git prend la main et fait une fusion tout seul, donc probabilité non nulle de corrompre les données.
2. Si tu travaille sur une branche, tu dois faire des fusions régulièrement pour avoir les dernières mise à jour des bugs. C'est déjà assez difficile d'avoir un historique propre juste pour le code, pas la peine d'en rajouter.
3. Potentiellement, tu dois changer de branche pour accéder à une version plus récente ou parallèle des bugs, donc mettre tes changements en cours dans une stash, checkout, lecture, et retour dans l'autre sens. C'est ni efficace ni simple.
4. Avoir les données des bugs à part ouvre beaucoup de possibilité, comme par exemple d'avoir des bugs indépendants les un des autres pour un checkout partiel, avoir une gestion des identités propres, etc, etc …
5. C'est plus facile de communiquer avec d'autres tracker quand le modèle de donnée n'est pas trop éloigné.

Mais tout ça, ça ne veut pas dire que les bugs ne peuvent pas être "conscient" de la notion de branche. C'est bien plus simple et souple de rajouter ce concept par la suite plutôt que de se lier les mains dans le dos tout de suite.

Pourquoi que les bugs?

Posté par cosmocat le 07/12/18 à 12:46. Évalué à 2 (+0/-0).

J'aime beaucoup l'idée depuis que j'avais découvert le plus maintenu **http://www.bugseverywhere.org/**

La question que je me pose, c'est pourquoi s'être restreint aux bugs, qui ne sont en fait qu'un cas particulier de tâches.

Comme ça, on pourrait gérer des taches et mettre ça sur un board…

J'avais découvert çà également, qui est pas mal sur le principe mais le format est mal fichu et, même s'il peut être commité, ne facilite pas le travail a plusieurs (et les merges)
https://marketplace.visualstudio.com/items?itemName=mkloubert.vscode-kanban

Génération du token

Posté par Steap le 07/12/18 à 12:52. Évalué à 0 (+0/-0).

> *La nouvelle version 0.4 apporte une fonctionalité qui a été beaucoup demandée, un importeur incrémental pour Github ! Après avoir utilisé git bug bridge configure pour saisir ses identifiants, un token d'API est généré automatiquement et git-bug devient capable d'aspirer les issues en local dans son modèle de données.*

Puis-je récupérer les bugs anonymement ? Puis-je fournir mon propre token plutôt que de taper mes identifiants ? Comment met-on à jour la liste des bugs (y a-t-il une commande similaire à "git fetch" ?) ?