[HOME](#)[FORMATIONS](#)[À PROPOS](#)

05 novembre 2018 11 commentaires

# Git: bien nommer ses commits

#Outils #Réflexions



Presque tous les développeurs utilisent aujourd'hui Git comme système de versionning. Cependant, au delà des **commandes à connaître**, un usage efficace passe par des messages de commit clairs et concis.

Néanmoins, même avec d'importants efforts, nommer ses commits n'est pas chose facile.

Au programme

1. **Le commit parfait**
2. **Souriez**

	COMMENT	DATE
○	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
○	ENABLED CONFIG FILE PARSING	9 HOURS AGO
○	MISC BUGFIXES	5 HOURS AGO
○	CODE ADDITIONS/EDITS	4 HOURS AGO
○	MORE CODE	4 HOURS AGO
○	HERE HAVE CODE	4 HOURS AGO
○	AAAAAAA	3 HOURS AGO
○	ADKFJSLKDFJSDKLEJ	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS	2 HOURS AGO
○	HAHAHAHAANDS	2 HOURS AGO
AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.		

**XKCD** illustre parfaitement ce à quoi finissent bien souvent par ressembler les repos git

Dans de telles conditions, on finit par ne plus trop s'y retrouver. Dès lors, l'usage de `git log`, `git blame` et autres perdent vite de leur intérêt.

Pour remédier à cela, de la même manière que de très nombreux projets ont des *style guides*, certains projets se dotent de normes afin d'imposer un style uniforme au messages de commit.

C'est par exemple le cas pour **Angular**. Le projet dispose de règles très claires qui peuvent convenir à tous types de dépôt. En se basant sur les règles d'Angular, nous allons voir ce qui constitue un bon message de commit et la forme qu'il doit prendre.

## Le commit parfait

Un bon message de commit doit permettre de savoir ce qui a changé et pourquoi. Le comment, c'est à dire la manière d'effectuer ces changements, n'a pas à être expliqué. La lecture du code et la mise en évidence des changements via un `diff` est explicite en soi.

Voici le format adopté par Angular.

```
<type>(<portée>): <sujet>
```

```
<description>
```

```
<footer>
```

On voit ici les différentes parties d'un commit et ce qu'elles doivent comporter. Voyons ces éléments dans le détail.

La première ligne comporte trois éléments : le type, la portée et le sujet. Voyons en quoi ils consistent.

## Type du commit

Le type nous informe du type de commit. 9 types sont disponibles :

- `build` : changements qui affectent le système de build ou des dépendances externes (npm, make...)
- `ci` : changements concernant les fichiers et scripts d'intégration ou de configuration (Travis, Ansible, BrowserStack...)
- `feat` : ajout d'une nouvelle fonctionnalité
- `fix` : correction d'un bug
- `perf` : amélioration des performances
- `refactor` : modification qui n'apporte ni nouvelle fonctionnalité ni d'amélioration de performances
- `style` : changement qui n'apporte aucune alteration fonctionnelle ou sémantique (indentation, mise en forme, ajout d'espace, renommage d'une variable...)
- `docs` : rédaction ou mise à jour de documentation
- `test` : ajout ou modification de tests

À cela s'ajoute `revert`. Ce dernier permet comme son nom l'indique, d'annuler un précédent commit. Dans ce cas, le message prend la forme suivante :

```
revert sujet du commit annulé hash du commit annulé
```

## La partie affectée (scope)


C'est la deuxième élément de la première ligne. Il nous permet immédiatement de savoir quelle partie du projet est affectée. Par exemple pour un site de e-commerce, on pourrait avoir *product*, *cart* ou *checkout*.

Cet élément est facultatif. En effet, il n'est parfois pas pertinent.

## Le sujet

Le sujet contient une description **succinte** des changements. En général, on se limite à 50 caractères. De nombreux outils avertissent d'ailleurs lorsque l'on dépasse la longueur maximale.

### Commit changes

 **ProTip!** Great commit summaries contain fewer than 50 characters. Place extra information in the extended description.

:docs: add a more detailed exemple to show how the |

Add an optional extended description...

Github prévient l'utilisateur si son sujet est trop long

Pour adopter un style descriptif efficace, on utilise l'impératif présent : *add*, *change*, *update*, *remove* et non pas *changed* ou *removed*. `add caching for better performance` par exemple.

Personnellement, je suis le style guide et je ne mets pas de majuscule à la première lettre du sujet ni ne met de point à la fin. L'essentiel est surtout d'adopter une politique et de s'y tenir afin d'être cohérent.

## Le corps du message

Beaucoup l'ignorent, mais les messages peuvent comporter un corps dans lequel on peut expliquer plus en détails la raison des changements. De même que pour le sujet, on utilisera l'impératif présent.

De nouveau, on explique ici la raison du changement et en quoi c'est nouvelle manière est différente de l'état précédent.

Le comment est visible directement dans le code. Par ailleurs, si le code est complexe, c'est le moment de penser à le commenter si ce n'est pas déjà fait !












## Le footer du commit

De même que le corps du message, le footer est facultatif. On l'utilisera d'ailleurs moins souvent.

On réserve le footer aux *breaking changes* et on y référence aussi le ticket d'erreur que règlent les modifications le cas échéant.

## Souriez

On voit de plus en plus l'usage d'emojis dans les commits. **Gitmoji** est un projet qui vise à définir un style guide quant à l'usage des emojis dans les commit.

Buzut  update docs		Latest commit ea1657c 2 days ago
 <code>.gitignore</code>	 (git) add gitignore	25 days ago
 <code>LICENCE.md</code>	 add licence	25 days ago
 <code>README.md</code>	 update docs	2 days ago
 <code>commit-msg</code>	 change icon for revert for a more expressive one	23 days ago
 <code>prepare-commit-msg</code>	 add exec rights to hooks (won't work otherwise)	25 days ago

Un des mes dépôts Github où j'utilise des emojis

Les emojis permettent de rapidement et visuellement transmettre du sens. De plus, un peu de fun dans les messages ne peut pas faire de mal.

En revanche, l'usage d'une palette d'emojis trop étendue dans les sujets peut mener à de la confusion car il n'est pas toujours évident que 🚀 a trait au déploiement.

Je n'utilise donc qu'un nombre d'icônes restreint qui correspondent aux différents types de commits.

Par ailleurs, je privilégie l'usage des emojis au format Unicode plutôt que d'utiliser les codes markdown. Ainsi, les emojis sont affichés dans tous les environnements qui supportent Unicode – ce qui est beaucoup plus répandu que les environnements qui supportent le markdown.

Pour me faciliter la tâche, j'utilise des hooks git afin de remplacer automatiquement le code du type par l'emoji correspondant. Ainsi, lorsque je saisis mon message de commit, il est automatiquement transformé. Je tape `:perf:(database) add caching for better performance` et le hook modifie mon message en ⚡(database) add caching for better performance.

Étant donné que les hooks ne font pas partie du code commité, ils ne sont pas automatiquement mis en place lors d'un `git clone`.

Aussi, pour des projets collaboratif, on peut facilement ajouter l'installation des hooks au système de build. `npm install` peut par exemple tout à fait inclure l'ajout des hooks.

Par ailleurs, j'ai un alias dans mon bash afin de pouvoir ajouter les hooks à un repo existant d'une simple commande.

J'ai publié les hooks avec toutes les explications sur [Github](#). N'hésitez donc pas à les inclure à vos projets !

## Commentaires



da dit – 5 novembre 2018

refactor\* à la place de refacor peut-être ;)

Répondre



Buzut dit – 5 novembre 2018

Ah oui en effet, c'est plus approprié :p Merci pour le signalement !

Répondre



Cascador dit – 5 novembre 2018

Yo,

Il manque pas une image par rapport à "Voici à quoi finissent bien souvent par ressembler les repos git" ?

Tcho !


Répondre



Buzut dit – 5 novembre 2018

Bien vu ! Une petite erreur de nom d'image mais comme l'ancien nom était dans mon cache je n'avais pas remarqué son absence. Merci :)

Répondre

 bob dit – 5 novembre 2018

Je suis développeur depuis 25 ans et je dois avouer que les gestionnaires de version git et compagnie ça me casse les couilles au même titre que les tests unitaires et toutes ces modes de branleurs...

Répondre



Buzut dit – 5 novembre 2018

Sur certains petits projets je n'en n'utilise pas. Mais sans ces trucs de branleurs comme tu dis, tu n'aurais pas la même qualité d'OS pour écrire ton commentaire ;)

Répondre

 Apitronix dit – 7 novembre 2018

Je comprends pas l'intérêt du type de commit "revert". Je veux dire... On peut déjà supprimer des commits distants avec Git, alors pourquoi faire un commit pour en supprimer un autre ?

Heureusement que tout le monde ici n'a pas 25 ans d'expérience, sinon qu'est-ce qu'on dirait d'âneries... :p

Signé : Un développeur débutant

Répondre



Buzut dit – 8 novembre 2018

En fait, ce n'est pas parce que tu peux que tu dois ! Quand tu travailles en collaboration sur une branche, c'est une mauvaise pratique de modifier l'historique. En effet, d'autres développeurs peuvent avoir cloné des changements déjà effectués, si entre temps l'historique de la remote change, ça devient assez compliqué...

Donc de manière générale, revert & co, ne sont à utiliser que sur des commits qui n'ont pas encore été pushés.

Répondre

 FAb dit – 18 novembre 2018

Merci pour l'article. J'avais lu d'autres façons de faire mais celle-ci est pas mal... Je vais voir si on peut l'adopter.

Sinon, cela ne ferait pas de mal dd'indiquer la source de l'image, surtout que c'est un super site : <https://xkcd.com/1296/>

Répondre



Buzut dit – 18 novembre 2018

Tu as raison pour xkcd, c'est ajouté ! Si tu as des feedbacks après la mise en place, je serai heureux que tu nous en fasses part en commentaires.

Répondre

 Httqm dit – 20 novembre 2018

"pourquoi faire un commit pour en supprimer un autre ?"

@Aptronix : Buzut a déjà répondu en partie à la question : il ne faut pas (non, il ne faut PAS !) réécrire l'historique d'un dépôt Git. L'autre raison de faire un commit pour en supprimer un autre, ce que cette suppression, comme tout changement apporté au code, a un auteur, une date et une raison d'être. Et lier ces infos à un changement (quel qu'il soit), c'est... un commit ;-)

Répondre

## Rejoignez la discussion !

Ce que vous voulez partager avec nous

Votre message

Vous pouvez utiliser Markdown pour les liens [ancree de lien](url), la mise en *italique* et en **gras**. Enfin pour le code, vous pouvez utiliser la syntaxe `inline` et la syntaxe bloc

```
```
```

```
ceci est un bloc  
de code  
```
```

Votre prénom ou pseudo

Pseudo

Votre email (un email vous sera envoyé pour valider le commentaire)

Email

Soumettre le commentaire

---

Quentin Busuttil – Licence Creative Commons BY-NC-ND