Git happens! 6 типових помилок Git і як їх виправити

22.06.2019 Cinex ІТ технології 0

***Прим. перек.**: Днями в блозі для інженерів улюбленого нами проекту GitLab з'явилася невелика, але дуже корисна замітка з інструкціями, які допомагають зберегти час і нерви в разі різних проблем, що трапляються в міру роботи з Git. Навряд чи вони будуть нові для досвідчених користувачів,*

ПОШУК

ПОШУК …

ПОЗНАЧКИ

APACHE AUTORUN BASH BITCOIN BLOCKCHAIN COMPOSER CRYPTOCURRENCY CSS DATABASE DJANGO DOCKER ELOQUENT ENGLISH ENUM GAMES GIMP GIT HTML JAVA JAVASCRIPT JQUERY LARAVEL MERCURIAL PARSING PHOTOSHOP PHP PHPSTORM

але обов'язково знайдуться і ті, кому вони знадобляться. А в кінець цього матеріалу ми додали невеликий бонус від себе. Доброю всім п'ятниці!

Всі ми робимо помилки, особливо при роботі з такими складними системами, як Git. Але пам'ятайте: Git happens!

Якщо ви тільки починаєте шлях з Git, навчитеся основам роботи з ним в командному рядку. А тут я розповім про те, як можна виправити шість найбільш поширених помилок в Git.

1. Упс ... Я помилився в повідомленні до останнього коміту

Після декількох годин кодингу легко припуститися помилки в повідомленні комітів. На щастя, це легко виправити:

```
1 git commit --amend
```

З цією командою відкриється текстовий редактор і дозволить внести зміни в повідомлення до останнього коміту. І ніхто не дізнається, що ви написали «addded» з трьома «d».

2. Упс ... Я забув додати файл до останнього коміту

Інша популярна помилка в Git - занадто поспішний коміт. Ви забули додати файл, забули його зберегти або повинні внести невелику зміну, щоб коміт став осмисленим? Вашим другом знову буде `--amend`.

Додайте відсутній файл і виконайте цю вірну команду:

```
1 git add missed-file.txt
```

PLAYONLINUX PLUGINS
PROBLEM SOLVING PYTHON REGEXP
SEO SOFTWARE ENGINEERING SPRING
SQL TELEGRAM TIPS AND TRICKS
TIPS AND TRICKS UNIT TESTING
WORDPRESS ВІРШІ ШПАРГАЛКИ

ОСТАННІ НОТАТКИ

20 Laravel Eloquent порад і трюків
15.12.2019

Як зробити скріншот сайту по URL на PHP
04.08.2019

```
git commit --amend
```

Тепер ви можете або відкоригувати повідомлення, або просто зберегти його в колишньому вигляді (з доданим файлом).

3. Упс ... Я додав файл, який не повинен бути в цьому репозиторії

Але що, якщо у вас зворотна ситуація? Що, якщо ви додали файл, який не хочете комітити? Оманливий ENV-файл, директорію збірки або фото з котом, що було випадково збережено в неправильному каталозі ... Все можна вирішити.

Якщо ви зробили тільки *stage* для файлу і ще не закомітили його, все робиться через простий *reset* потрібного файлу (що знаходиться в stage):

```
git reset /assets/img/misty-and-pepper.jpg
```

Якщо ж ви все-таки закомітили зміни, буде потрібно додатковий підготовчий етап:

```
git reset --soft HEAD~1
git reset /assets/img/misty-and-pepper.jpg
rm /assets/img/misty-and-pepper.jpg
git commit
```

Коміт буде відвалений, картинка видалена, а потім зроблений новий коміт.

***Прим. перек.**: Як відмічено в коментарях до оригінальної статті, цю проблему теж можна вирішити за допомогою вже згаданого `--amend`. По всій видимості, цим пунктом автор хотів показати, які ще є способи зміни історії комітів для виправлення помилки.*

4. Упс ... Я закомітив зміни в master

Docker Cheat Sheet
05.07.2019

Гарячі клавіші Ubuntu Linux
01.07.2019

Git happens! 6 типових помилок Git і як їх виправити
22.06.2019

НЕДАВНІ КОМЕНТАРІ

Отже, ви працюєте над новою фічею і поспішили, забувши створити нову гілку для неї. Ви вже закомітили купу файлів і всі ці коміти виявилися в master'і. На щастя, GitLab може запобігати push'ам прямо в master. Тому ми можемо відкотити всі потрібні зміни в нову гілку наступними трьома командами:

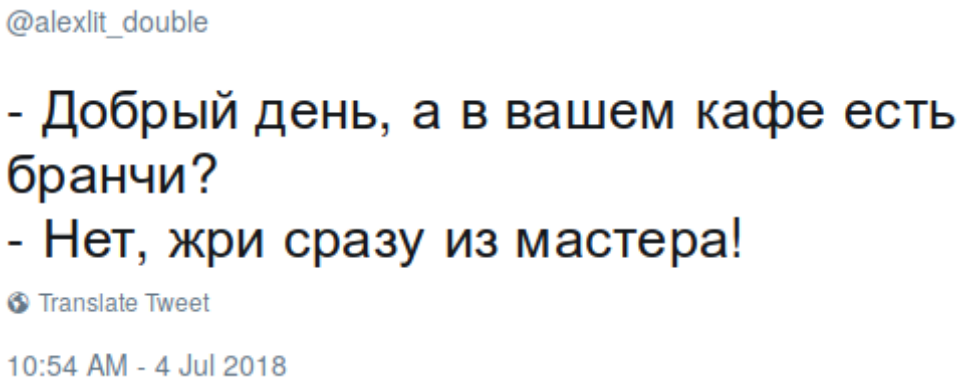
***Примітка**: Переконайтеся, що спочатку закомітили або stash'нули свої зміни - інакше всі вони будуть втрачені!*

```
1 git branch future-brunch
2 git reset HEAD~ --hard
3 git checkout future-brunch
```

Буде створена нова гілка, в master'і - проведений відкат до стану, в якому він був до ваших змін, а потім зроблений *checkout* нової гілки з усіма вашими змінами.

5. Упс ... Я зробив помилку в назві гілки

Найбільш уважні могли помітити в попередньому прикладі помилку в назві гілки. Уже майже 15:00, а я все ще не обідав, тому мій голод назвав нову гілку (*br**a**nch*) як *future-br**u**nch*. Смакота!



Перейменуємо цю гілку аналогічним способом, що використовується при перейменуванні файлу за допомогою команди *mv*, тобто помістивши її в нове місце з правильною назвою:

```
git branch -m future-brunch feature-branch
```

Якщо ви вже push'нули цю гілку, знадобиться пара додаткових кроків. Ми видалимо стару гілку з *remote* і push'нем нову:

```
git push origin --delete future-brunch
git push origin feature-branch
```

***Прим. перек.**: Видалити гілку з remote ще можна за допомогою:*

```
git push origin :future-brunch
```

6. Oops... I did it again!

Остання команда на той випадок, коли все пішло не так. Коли ви накопіювали і навставляли купу рішень з Stack Overflow, після чого в репозиторії все стало ще гірше, ніж було на початку. Всі ми одного разу стикалися з подібним ...

`git reflog` показує список всіх виконаних вами операцій. Потім він дозволяє використовувати магічні можливості Git'а по подорожі в часі, тобто повернутися до будь-якого моменту з минулого. Мушу зазначити, що це ваша остання надія - не варто вдаватися до неї в простих випадках. Отже, щоб отримати список, виконайте:

```
git reflog
```

Кожен наш крок знаходиться під чуйним наглядом Git'а. Запуск команди на проекті вище видав наступне:

```
3ff8691 (HEAD -> feature-branch) HEAD@{0}: Branch: renamed refs/heads/future-brunch to r
3ff8691 (HEAD -> feature-branch) HEAD@{2}: checkout: moving from master to future-brunch
2b7e508 (master) HEAD@{3}: reset: moving to HEAD~
3ff8691 (HEAD -> feature-branch) HEAD@{4}: commit: Adds the client logo
2b7e508 (master) HEAD@{5}: reset: moving to HEAD~1
37a632d HEAD@{6}: commit: Adds the client logo to the project
2b7e508 (master) HEAD@{7}: reset: moving to HEAD
2b7e508 (master) HEAD@{8}: commit (amend): Added contributing info to the site
dfa27a2 HEAD@{9}: reset: moving to HEAD
dfa27a2 HEAD@{10}: commit (amend): Added contributing info to the site
700d0b5 HEAD@{11}: commit: Addded contributing info to the site
efba795 HEAD@{12}: commit (initial): Initial commit
```

Зверніть увагу на самий лівий стовпець - це індекс. Якщо ви хочете повернутися до будь-якого моменту в історії, виконайте наступну команду, замінивши `{index}` на відповідне значення (наприклад, `dfa27a2`):

```
git reset HEAD@{index}
```

Отже, тепер у вас є шість способів вибратися з найчастіших Gitfalls *(ігра слів: pitfall перекладається як «пастка, помилка» — **прим. перек.**)*.

Бонус від перекладача

По-перше, цінне зауваження до всього написаного вище (крім пункту 5). Потрібно враховувати, що ці дії змінюють історію комітів, тому їх слід проводити, тільки якщо зміни не були відправлені в remote (push'нути). В іншому випадку старий поганий коміт вже буде на remote-гілці і доведеться або виконувати `git pull` (который зробить *merge*, і тоді спроба «очистити» історію призведе до гірших наслідків), або `git push --force`, що може призвести до втрати даних при роботі з гілкою кількох людей...

Тепер - невеликі корисні додатки з нашого досвіду:

- Якщо ви (випадково чи ні) змінили гілку і вам потрібно повернутися на попередню, найшвидший спосіб - використовувати `git checkout -`.
- Якщо ви випадково додали до коміту файл, який не повинен бути туди доданий, але ще не зробили коміт - використовуйте `git reset HEAD path/to/file`. Схожа ситуація описана в пункті 3, але в дійсності вона ширше, тому що відноситься до будь-яких непотрібним змін в коміту (не тільки до випадку зайвого файлу).
- Доброю практикою, щоб не закомітити зайвого, є використання параметра **-p** при додаванні файлу до коміту (`git add -p`). Це дозволяє зробити review кожної зміни, який піде в коміт. Але варто пам'ятати, що він не додає до коммітів untracked-файли - їх потрібно додавати без цього параметра.
- Ряд хороших рекомендацій (в тому числі і більш складних), можна знайти в статті 2014 року «Git Tutorial: 10 Common Git Problems and How to Fix Them». Зокрема, зверніть увагу на використання `git revert` і `git rebase -i`.

Джерело

GIT TIPS AND TRICKS

« ПОПЕРЕДНІЙ
Новий синтаксис JSON-стовпця where() і update() в Laravel 5.3

ДАЛІ »
Гарячі клавіші Ubuntu Linux

ЗАЛИШТЕ ПЕРШИЙ КОМЕНТАР

Залишити коментар

Вашу адресу електронної пошти не буде опубліковано

Коментувати

Ім'я*

Email *

Збережіть моє ім'я, електронну пошту у цьому веб-переглядачі наступного разу, коли я коментуватиму.

ОПУБЛІКУВАТИ КОМЕНТАР