



Флант

Специалисты по DevOps и Kubernetes



gsmetal 10 августа 2018 в 11:03

Git happens! 6 типичных ошибок Git и как их исправить

Автор оригинала: Sam Beckham

Блог компании Флант, Git, Системы управления версиями

Перевод



Прим. перев.: На днях в блоге для инженеров любимого нами проекта [GitLab](#) появилась небольшая, но весьма полезная заметка с инструкциями, которые помогают сохранить время и нервы в случае различных проблем, случающихся по мере работы с Git. Вряд ли они будут новы для опытных пользователей, но обязательно найдутся и те, кому они пригодятся. А в конце этого материала мы добавили небольшой бонус от себя. Хорошей всем пятницы!

Все мы делаем ошибки, особенно при работе с такими сложными системами, как Git. Но помните: Git happens!

Если вы только начинаете путь с Git, обучитесь основам работы с ним в командной строке. А здесь я расскажу о том, как можно исправить шесть наиболее распространённых ошибок в Git.

1. Упс... Я ошибся в сообщении к последнему коммиту

После нескольких часов кодинга легко допустить ошибку в сообщении коммита. К счастью, это легко исправить:

```
git commit --amend
```

С этой командой откроется текстовый редактор и позволит внести изменения в сообщение к последнему коммиту. И никто не узнает, что вы написали «addded» с тремя «d».

2. Упс... Я забыл добавить файл к последнему коммиту

Другая популярная ошибка в Git — слишком поспешный коммит. Вы забыли добавить файл, забыли его сохранить или должны внести небольшое изменение, чтобы коммит стал осмысленным? Вашим другом снова будет `--amend`.

Добавьте недостающий файл и выполните эту верную команду:

```
git add missed-file.txt
git commit --amend
```

Теперь вы можете либо откорректировать сообщение, либо просто сохранить его в прежнем виде (с добавленным файлом).

3. Упс... Я добавил файл, который не должен быть в этом репозитории

Но что, если у вас обратная ситуация? Что, если вы добавили файл, который не хотите коммитить? Обманчивый ENV-файл, директорию сборки или фото с котом, что было случайно сохранено в неправильном каталоге... Всё решаемо.

Если вы сделали только `stage` для файла и ещё не коммитнули его, всё делается через простой `reset` нужного файла (находящегося в `stage`):

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

Если же вы всё-таки коммитнули изменение, потребуется дополнительный предварительный шаг:

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

Коммит будет откатчен, картинка удалена, а затем сделан новый коммит.

Прим. перев.: Как замечено в комментариях к оригинальной статье, эту проблему тоже можно решить с помощью уже упомянутого `--amend`. По всей видимости, данным пунктом автор хотел показать, какие ещё есть способы изменения истории коммитов для исправления ошибки.

4. Упс... Я коммитнул изменения в master

Итак, вы работаете над новой фичей и поспешили, забыв создать новую ветку для неё. Вы уже коммитнули кучу файлов и все эти коммиты оказались в `master`'е. К счастью, [GitLab](#) может предотвращать `push`'ы прямо в `master`. Поэтому мы можем откатить все нужные изменения в новую ветку следующими тремя командами:

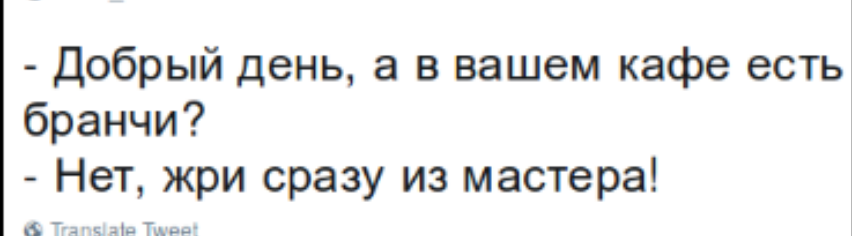
Примечание: Убедитесь, что сначала коммитнули или `stash`'нули свои изменения — иначе все они будут утеряны!

```
git branch future-branch
git reset HEAD~ --hard
git checkout future-branch
```

Будет создана новая ветка, в `master`'е — произведён откат до состояния, в котором он был до ваших изменений, а затем сделан `checkout` новой ветки со всеми вашими изменениями.

5. Упс... Я сделал ошибку в названии ветки

Самые внимательные могли заметить в предыдущем примере ошибку в названии ветки. Уже почти 15:00, а я всё ещё не обещал, поэтому мой голод назвал новую ветку (`branch`) как `future-branch`. Вкуснотища!



Переименуем эту ветку аналогичным способом, что используется при переименовании файла с помощью команды `mv`, то есть поместив её в новое место с правильным названием:

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

Если вы уже `push`'нули эту ветку, понадобится пара дополнительных шагов. Мы удалим старую ветку из `remote` и `push`'нем новую:

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

Прим. перев.: Удалить ветку из `remote` ещё можно с помощью:

```
git push origin :future-branch
```

6. Oops... I did it again!

Последняя команда на тот случай, когда всё пошло не так. Когда вы накопили и навставляли кучу решений со [Stack Overflow](#), после чего в репозитории всё стало ещё хуже, чем было в начале. Все мы однажды сталкивались с подобным...

`git reflog` показывает список всех выполненных вами операций. Затем он позволяет использовать магические возможности Git'a по путешествию во времени, т.е. вернуться к любому моменту из прошлого. Должен отметить, что это ваша последняя надежда — не стоит прибегать к ней в простых случаях. Итак, чтобы получить список, выполните:

```
git reflog
```

Каждый наш шаг находится под чутким наблюдением Git'a. Запуск команды на проекте выше выдал следующее:

```
3ff8691 (HEAD -> feature-branch) HEAD@{0}: Branch: renamed refs/heads/future-branch to refs/heads/feature-branch
3ff8691 (HEAD -> feature-branch) HEAD@{2}: checkout: moving from master to future-branch
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 (master) 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: Added contributing info to the site
efba795 HEAD@{12}: commit (initial): Initial commit
```

Обратите внимание на самый левый столбец — это индекс. Если вы хотите вернуться к любому моменту в истории, выполните следующую команду, заменив `{index}` на соответствующее значение (например, `dfa27a2`):

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

Итак, теперь у вас есть шесть способов выбраться из самых частых Gitfalls (*игра слов: `pitfall` переводится как «ловушка, ошибка» — прим. перев.*).

Бонус от переводчика

Во-первых, ценное замечание ко всему написанному выше (кроме пункта 5). Нужно учитывать, что эти действия меняют историю коммитов, поэтому их следует проводить, только если изменения не были отправлены в `remote` (`push`'нуты). В противном случае старый плохой коммит уже будет на `remote`-ветке и придётся либо выполнять `git pull` (который делает `merge`, и тогда попытка «почистить» историю приведёт к худшим последствиям), либо `git push --force`, что чревато потерей данных при работе с веткой нескольких человек...



Теперь — небольшие полезные дополнения из нашего опыта:

- Если вы (случайно или нет) сменили ветку и вам нужно вернуться на предыдущую, самый быстрый способ — использовать `git checkout -`.
- Если вы случайно добавили к коммиту файл, который не должен быть туда добавлен, но ещё не сделали коммит — используйте `git reset HEAD path/to/file`. Похожая ситуация описана в пункте 3, но в действительности она шире, т.к. относится к любым ненужным изменениям в коммите (не только к случаю лишнего файла).
- Хорошей практикой, чтобы не закомитить лишнего, является использование параметра `-p` при добавлении файла к коммиту (`git add -p`). Это позволяет сделать `review` каждого изменения, которое уйдёт в коммит. Но стоит помнить, что он не добавляет к коммиту `untracked`-файлы — их нужно добавлять без этого параметра.
- Ряд хороших рекомендаций (в том числе и более сложных), можно найти в статье 2014 года «[Git Tutorial: 10 Common Git Problems and How to Fix Them](#)». В частности, обратите внимание на использование `git revert` и `git rebase -i`.

P.S. от переводчика

Читайте также в нашем блоге:

- «GitLab CI для непрерывной интеграции и доставки в production»: Часть 1 (наш пайплайн); Часть 2 (преодолевая трудности);
- «Лучшие практики CI/CD с Kubernetes и GitLab (обзор и видео доклада)»;
- «Сборка проектов с GitLab CI: один .gitlab-ci.yml для сотни приложений».

Теги: Git

Хэбы: Блог компании Флант, Git, Системы управления версиями


+61

601


99,8k

62

Поделиться



Сергей @gsmetal
Разработчик



Флант
Специалисты по DevOps и Kubernetes

Сайт

Facebook

Twitter

ВКонтакте

Github

