如何將既有 repo 無痛轉移到新 repo 並保持 commit 歷史紀錄

https://medium.aifian.com/%E5%A6%82%E4%BD%95%E5%B0%87%E6%97%A2%E6%9C%89-repo-%E7%84%A1%E7%97%9B%E8%BD%89%E7%A7%BB%E5%88%B0%E6%96%B0-repo-%E4%B8%A6%E4%BF%9D%E6%8C%81-commit-%E6%AD%B7%E5%8F%B2%E7%B4%80%E9%8C%84-39d1c282bf44
May 31 · 10 min read

當團隊正準備建立一個新專案時，卻還未決定關鍵性的方法，例如：用什麼 UI 套件、用哪一個 CSS 預處理 (CSS preprocessor) 器等。這時筆者先開啟一個專案，去嘗試哪一些套件比較接近想達成的目的，甚至是符合團隊的開發習慣，最後實驗出最適合團隊的選擇後，把原本為實驗性質的 repository 從個人的 Git 存放庫 (例如 GitHub, GitLab, Bitbucket 等) 轉移到公司底下。但我們還想保留所有的 commit 歷史紀錄，好讓未來可以繼續追蹤，以及透過 commit message 知道當時下的註解，那這時應該怎麼做呢？

這篇文章將介紹如何將一個 repository 轉移到另一個 repository 並且還保留它原有的歷史紀錄。

首先我們想在遵循兩個原則的前提下去完成轉移：

1. **不對原始 repository 產生額外影響**
2. **保留原有的 commit 歷史紀錄**

以下為了閱讀順暢會將 repository 簡稱為 repo。

實際上我們可能會遇到兩個不同的情境，第一個情境是：

新的 repo 沒有任何記錄、完全是空的 (解法一)

意思是有一個空的 repo 裡面沒有任何檔案，也沒有任何 commit 紀錄。文章中將介紹適用於此情境的兩種解法，第一種所使用到的 Git 指令較平易近人，在開發中頻繁被應用，相信時常使用 Git 的讀者一定不陌生；第二種則是相對少見的 Git 指令 (git --mirror)，將在文末做介紹，讓我們先來看看第一種解法吧！

一、透過 git clone 方法來複製 repo 到一個新的資料夾

```
git clone [old-repo-url] new-repo-name
cd new-repo-name
```

這時候你用 git remote -v 檢查會發現，origin 的指向為 [old-repo-url]

二、用 remote set-url 將 origin remote 改成指向 [new-repo-url]

```
git remote set-url origin [new-repo-url]
```

三、push 指令後加上 --all, --tags 將 branches, tags 也 push 至新 repo，關於這兩個指令的詳情可以閱讀：https://git-scm.com/docs/git-push

```
git push --all
git push --tags
```

這樣我們就已經成功將整份舊 repo 轉移到新 repo 了。

新 repo 和舊 repo 的 commit 源頭不一樣

第二個情境是團隊遇到的經驗，在創建新 repo 的同時順手創了 .gitignore 或 README ，此時新的 repo 已經有了其他 commit， 也就是說這份新 repo 和舊 repo 有 commit 源頭不一樣的情況， 這時候我們要做的事情有些不同。

一、在新 repo 從 master branch 新增一個 main branch
這邊的有一個重點是：若新 repo 也有 master branch，就會跟舊 repo master branch 撞名，一旦撞名就不能直接切換 branch 過去，所以新 repo 設定的 branch 名稱不能跟舊 repo 的任何 branches 同名。 在此以新增一個 main branch 為例，如果你已經有名為 main 的 branch，請再改一個名稱。

```
git checkout master
git checkout -b main
```

二、在舊 repo 新增一個 remote 指向新 repo 這裡以 newRepo 代表該 remoto 名稱，實作時請隨意命名即可

```
git remote add newRepo [new-repo-url]
```

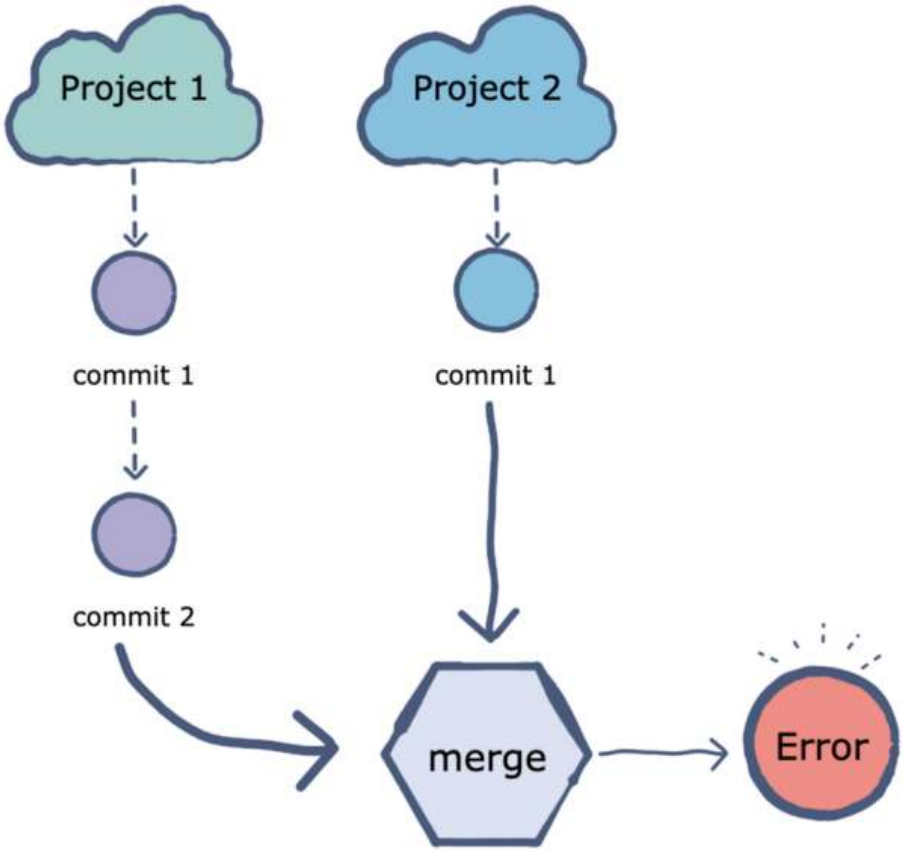
三、將 newRepo main branch 的程式碼抓下來，並切換到 main branch

```
git fetch newRepo main
git checkout newRepo/main
```

四、在 main branch 合併舊 repo 的 master branch

```
git merge master --allow-unrelated-histories
// 執行完後可能會遇到 conflict
```

如果你在這個步驟只有單純 merge 而沒有加上 --allow-unrelated-histories 的話，它會顯示拒絕合併沒有相關歷史紀錄的錯誤：“fatal: refusing to merge unrelated histories”，這是因為我們試圖合併兩個歷史紀錄毫不相干的 repo 所造成。



圖片出處：
https://www.educative.io/edpresso/the-fatal-refusing-to-merge-unrelated-histories-git-error

這邊利用到的 --allow-unrelated-histories 指令是在 Git 2.9.0 才有的變動：
https://github.com/git/git/blob/master/Documentation/RelNotes/2.9.0.txt#L58-L68

“git merge” used to allow merging two branches that have no common base by default, which led to a brand new history of an existing project created and then get pulled by an unsuspecting maintainer, which allowed an unnecessary parallel history merged into the existing project. The command has been taught **not to allow this by default,** *with an escape hatch “ –allow-unrelated-histories” option to be used in a rare event that merges histories of two projects that started their lives independently.*

git merge 的預設情況下是不允許兩個歷史紀錄不相關的 branches 做合併，所以才需要在後面加入這段指令。

五、解完 conflict 之後、就可以把程式碼推到 main branch 了

git push newRepo main

若要保留舊有 repo branches 就可以接著執行 **git push --all** 但其中名字相同的 branches (例如 master branch)會被擋下，原因如步驟一所提及。 以下附上執行後的部分結果：

```
 * [new branch]      issue#1 -> issue#1
 * [new branch]      issue#1-1 -> issue#1-1
 * [new branch]      issue#2 -> issue#2
 * [new branch]      issue#3 -> issue#3
 * [new branch]      issue#4 -> issue#4
 * [new branch]      issue#4-1 -> issue#4-1
 * [new branch]      issue#6 -> issue#6
 ! [rejected]        master -> master (fetch first)
```

上述介紹的指令都是大家熟悉的指令，只要瞭解 Git 版本控制和 repo 的本身歷史紀錄就很好理解脈絡。另外，想再分享一個筆者在查詢資料的過程中，發現的有趣指令：

– mirror Set up a mirror of the source repository. This implies – bare. Compared to – bare, – mirror not only maps local branches of the source to local branches of the target, it maps all refs (including remote-tracking branches, notes etc.) and sets up a refspec configuration such that all these refs are overwritten by a git remote update in the target repository.

git --mirror 可以幫我們直接複製一份一模一樣的 repo 並且包含所有的 refs (例如遠端追蹤的分支等)，而 refspec 設置讓未來只要透過 git remote update 指令就可以讓這份 repo 更新所有的 refs。

新的 repo 沒有任何記錄、完全是空的（解法二）

呼應到文章最一開始的第一個情境：要把一個 repo 轉移到一個裡面沒有任何檔案、也沒有任何 commit 紀錄的空 repo。實作利用 git --mirror 指令如何達到我們的目標：

一、clone 一份舊 repo 到本地端的新資料夾（名稱隨意）

git clone --mirror [old-repo-url] new-repo

二、移動到新資料夾並 push 至想存放 repo 的 url

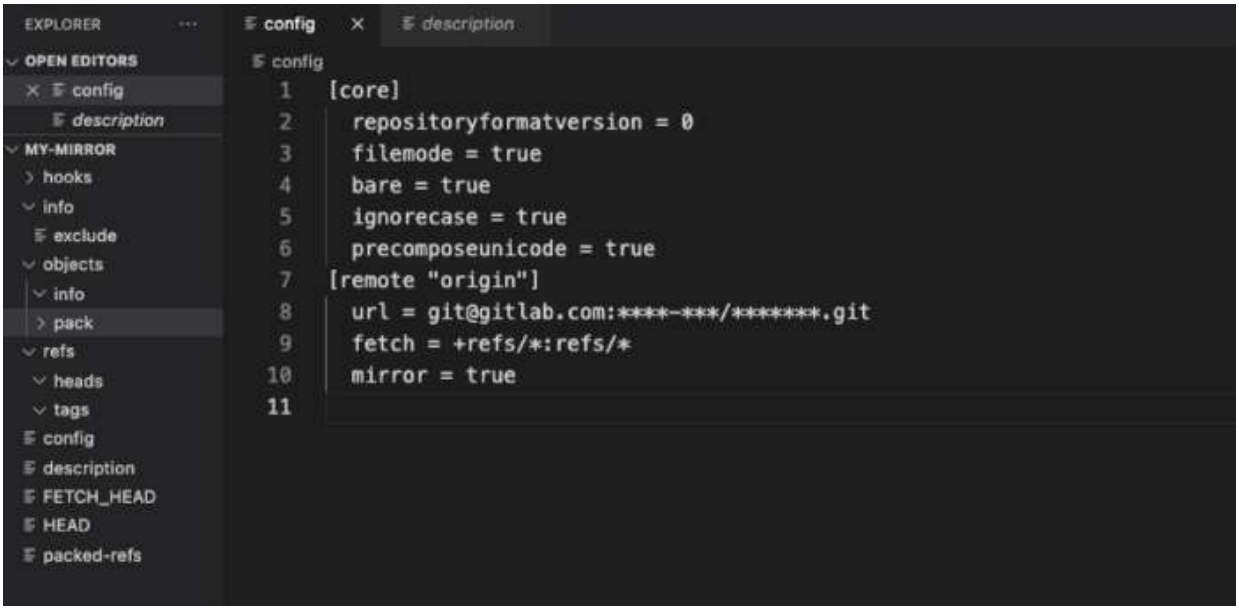
```
cd new-repo
git push --mirror [repo-url]
```

恭喜，此時你已經完成了移轉了！・

以下是 push 後結果的部分截圖，你會看到所有的 branches, refs 也被推上去了。

```
* [new branch]      issue#4 -> issue#4
* [new branch]      issue#4-1 -> issue#4-1
* [new branch]      issue#6 -> issue#6
* [new branch]      master -> master
* [new branch]      test -> test
* [new branch]      refs/merge-requests/1/head -> refs/merge-requests/1/head
* [new branch]      refs/merge-requests/1/merge -> refs/merge-requests/1/merge
* [new branch]      refs/merge-requests/2/head -> refs/merge-requests/2/head
* [new branch]      refs/merge-requests/2/merge -> refs/merge-requests/2/merge
```

看到這裡你可能會問 clone --mirror 指令下的產物是什麼，怎麼好像很厲害？跟大家分享一下，mirror repository 打開長這樣：



mirror repository 是不存在 “working directory” 的 repository，也就是說它不存在 source code，只有一些設定檔用來紀錄 commits, tags, branches, remote-tracking branches 等 Git 情報。

最後有個小提醒，如果你會利用 mirror repository 來維持兩個 repository 之間的一致性，那建議在移轉完成之後，將 mirror repository 的 remote origin 指向至新的 repo URL，預防之後工作繁忙，在 push 時不小心推到舊的 repo 。