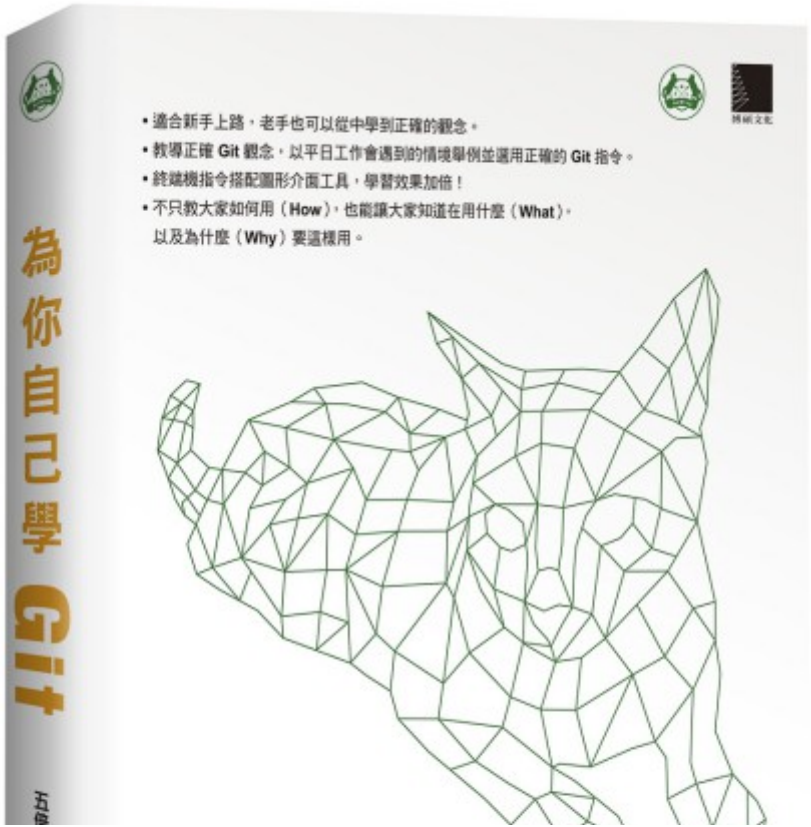
《為你自己學 Git》閱讀筆記

人生不能重來，但 GIT 可以（Git 篇）

Leo Kao Follow
Jan 29, 2018 · 15 min read

Git 是什麼？能吃嗎？（誤
礙於之前版控使用 Git 也只會基本指令操作（add / commit / push / pull …），前陣子也有去聽本書作者的演講覺得頗為清楚，這邊筆記一下學習內容。（需要更完整的資料請參照以下連結或買書吧～:D）
參考的是作者放在網上的資料：為你自己學 Git。並沒有全部章節都筆記，所以下面章節數對應內容章節數。



為你自己學 Git

. . .

四、設定 Git

使用者設定

設定 Name & Email

```
$ git config --global user.name "Leo Kao"
$ git config --global user.email "leokao0726@gmail.com"
$ git config --list    //檢視
user.name=Eddie Kao
user.email=eddie@5xruby.tw
```

設定檔位置預設會在自己帳號下的 `.gitconfig` 這個檔案裡。
檔案： `~/.gitconfig`

其他設定

設定縮寫，例如：

```
$ git config --global alias.co checkout
$ git config --global alias.br branch
$ git config --global alias.st status    //以 st 代替 status
```

. . .

五、開始使用 Git

新增、初始 Repository

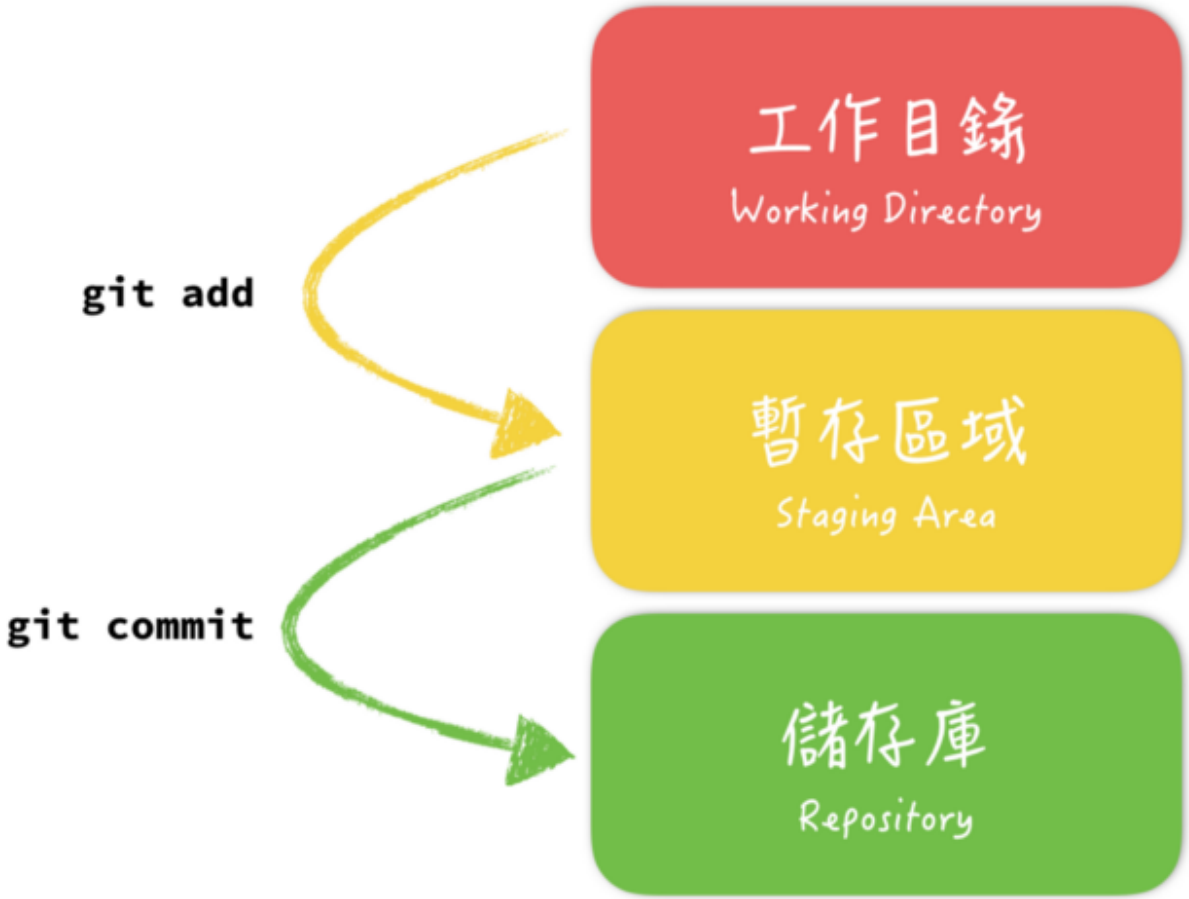
開始

```
$ git init                    # 初始化這個目錄，讓 Git 對這個目錄開始進行版控
Initialized empty Git repository in /private/tmp/git-practice/.git/
```

`git init` 會先建立一個 `.git` 目錄。
不想被 git 控制？移除 `.git` 即可。
為何使用 `/tmp` 目錄？MacOS 系列的作業系統， `/tmp` 目錄裡的東西在下次電腦重新開機（或當機）的時候就全部被清空，所以不需要再手動整理。

. . .

工作區、暫存區、儲存庫



> *指令*
> `git add` *把檔案從工作目錄移至暫存區（或索引）。*
> `git commit` *把暫存區的內容移至儲存庫。*

什麼時候要 Commit？

什麼時候都可以。常見的 Commit 時間點有：

1. 完成一個任務時：不管是大到完成一整個電子商務的金流系統，還是小至只加了一個頁面甚至只是改幾個字，都算是任務。
2. 下班時：雖然可能還沒完全搞定任務，但至少先 Commit 今天的進度，除了備份之外，也讓公司知道你今天有在努力工作。
3. 你想要 Commit 時。

複習

```
$ touch index.html                          # 建立檔案 index.html
$ git add index.html                        # 把 index.html 加至暫存區
$ git commit -m "create index page"         # 進行 Commit
```

. . .

檢視紀錄

```
$ git log
commit cef6e4017eb1a16a7bb3434f12d9008ff83a821a (HEAD -> master)
Author: Eddie Kao <eddie@5xruby.tw>
Date:   Wed Aug 2 03:02:37 2017 +0800

    create index page

commit cc797cdb7c7a337824a25075e0dbe0bc7c703a1e
Author: Eddie Kao <eddie@5xruby.tw>
Date:   Sun Jul 30 05:04:05 2017 +0800

    init commit
```

找某人的 Commit？

```
$ git log --oneline --author="Sherly"
```

. . .

[狀況] 修改 Commit 紀錄

修改 Commit 紀錄有好幾種方法：

1. 把 `.git` 目錄整個刪除（誤）。
2. 使用 `git rebase` 來修改歷史。
3. 先把 Commit 用 `git reset` 拆掉，整理後再重新 Commit。
4. 使用 `--amend` 參數來修改最後一次的 Commit。

這邊先筆記第 4 種。

使用 `--amend` 參數來進行 Commit

```
$ git log --oneline
4879515 WTF
$ git commit --amend -m "Welcome To Facebook"
$ git log --oneline
614a90c Welcome To Facebook
```

如果沒有加上 `-m` 參數提供要修改的訊息，就會跳出 Vim 視窗讓你編輯。再回來看紀錄，原本的就會被改成 “Welcome To Facebook” 。

> *雖然只是改紀錄的訊息，其它什麼都沒有改，但對 Git 來說因為「Commit 的內容」改變了，所以 Git 會重新計算並產生一顆新的 Commit 物件，也就是這其實是一次全新的 Commit（只是看起來不像新的）。*

修改更早的紀錄？使用 Rebase 指令來處理， `--amend` 參數只能處理最後一次而已。

> *雖然這只是改訊息，不管如何它就是修改了一次的歷史，所以請儘量不要在已經 Push 出去之後再修改，否則可能會造成其它人的困擾。*

. . .

[狀況] 追加檔案到最近一次的 Commit

> 剛剛完成 Commit，但發現有一個檔案忘了加到，又不想為了這個檔案重新再發一次 Commit… Q

兩種方法：

1. 使用 `git reset` 把最後一次的 Commit 拆掉，加入新檔案後再重新 Commit。
2. 使用 `--amend` 參數進行 Commit。

```
$ git commit --amend --no-edit
```

這樣就可以把檔案併入最後一次的 Commit。而最後面那個 `--no-edit` 參數的意思是指「我不要編輯 Commit 訊息」，所以就不會跳出 Vim 編輯器的視窗。

> *像這樣的修改歷史，請儘量不要使用在已經Push 出去的Commit 上。*

. . .

[狀況] 新增目錄

> *新增了一個images 目錄，但卻發現這個目錄好像沒辦法被加到Git 裡面？*

Git 在計算、產生物件的時候，是根據「檔案的內容」去做計算的，所以光是新增一個目錄，Git 是沒辦法處理它的。

> *注意！*空的目錄無法被提交！

. . .

[狀況] 有些檔案我不想放在 Git 裡

有些比較機密的檔案不想放在 Git 裡一起備份，例如資料庫的存取密碼或是 AWS 伺服器的存取金鑰...

只要在專案目錄裡放一個 `.gitignore` 檔案，並且設定想要忽略的規則就行了。如果這個檔案不存在，就手動新增它：

```
$ touch .gitignore
```

然後編輯

```
# 檔案名稱 .gitignore

# 忽略 secret.yml 檔案
secret.yml

# 忽略 config 目錄下的 database.yml 檔案
config/database.yml

# 忽略所有 db 目錄下附檔名是 .sqlite3 的檔案
```

```
# 當然你要忽略自己也可以，只是通常不會這麼做
# .gitignore
```

只要 `.gitignore` 這個檔案存在，即使這個檔案沒被 Commit 或是沒被 Push 上 Git Server 就會有效果。但這個檔案會建議 Commit 進專案並且推上 Git Server，這樣一來整個專案一起開發的人可以共享相同的設定。

. . .

[狀況] 檢視特定檔案的 Commit 紀錄

如果只想檢視單一檔案的紀錄，只要在 `git log` 後面接上那個檔名。

```
$ git log welcome.html
```

如果想看這個檔案到底每次的 Commit 做了什麼修改，可以再給它一個 `-p` 參數。

```
$ git log -p welcome.html
```

. . .

[狀況] 啊！不小心把檔案或目錄刪掉了...

如果被 rm 掉的檔案，可以：

```
$ git checkout cinderella.html  //救回單一檔案
$ git checkout .    //將所有檔案救回
$ ls -al  //查看檔案列表
```

怎樣都救得回來？

也不能這麼說，因為整個 Git 的紀錄都是放在根目錄的 `.git` 目錄裡，所以如果這個目錄被刪了，也就是表示歷史紀錄也被刪了，那就沒得救了。

[冷知識] Git 是去哪裡把檔案救回來的？

（Working Directory）的內容或檔案。所以當在上面執行 `git checkout welcome.html` 或 `git checkout .` 的時候，它會把 welcome.html 這個檔案，或是當下目錄所有檔案回復到上一次 Commit 的狀態。

```
$ git checkout HEAD~2 .
```


這個指令的意思就是「拿距離現在兩個版本以前的檔案來覆蓋現在工作目錄的檔案，同時也更新暫存區裡的狀態」。

. . .

[狀況] 剛才的 Commit 後悔，想拆掉重做...

如果 git 紀錄為：

```
$ git log --oneline
e12d8ef (HEAD -> master) add database.yml in config folder
85e7e30 add hello
657fce7 add container
abb4f43 update index page
cef6e40 create index page
cc797cd init commit
```

如果想拆掉最後一次的 Commit，可以使用「相對」或「絕對」的做法。「相對」的做法可以這樣：

```
$ git reset e12d8ef^     //每一個 ^ 符號表示「前一次」的意思
$ git reset master^
$ git reset HEAD^    // HEAD 跟 master 都在 e12d8ef 所以一樣結果
```

如果你很清楚想要把目前的狀態退回到哪個 Commit，可以直接指明：

```
$ git reset 85e7e30  //切換到 85e7e30 這個 Commit 的狀態
```

Commit 拆掉了，那拆出來的那些檔案跑哪去了？ Reset 。

`git reset` 指令可以搭配參數使用，常見到的三種參數，分別是 `--mixed（丟回工作目錄）` 、 `--soft（丟回暫存區）` 以及 `--hard（直接丟掉）` 。

> *$ git reset HEAD~2*
> *這個指令你原本可能會解讀成「請幫我拆掉最後兩次的Commit」，但其實用「拆」這個動詞只是我們比較容易理解而已，事實上並沒有真的把這個Commit「拆掉」（放心，所有的Commit 都還在）。*
> *正確的說，上面這個指令應該要解讀成「我要前往兩個Commit 之前的狀態」或是「我要變成兩個Commit 之前的狀態」，而隨著使用不同的參數模式，原本的這些檔案就會丟去不同的區域。*
> *因為實際上 git reset 指令也並不是真的刪除或是重新設定Commit，只是「前往」到指定的Commit，那些看起來好像不見的東西只是暫時看不到，但隨時都可以再撿回來。*

. . .

[狀況] 不小心使用 hard 模式 Reset 某個 Commit，救得回來嗎？

> *[觀念]不管是用什麼模式進行Reset，Commit 就是Commit，並不會因為你Reset 它然後就馬上消失了。*

如果想要退回剛剛 Reset 的這個步驟，只要 Reset 回到一開始那個 Commit 的 SHA-1 `e12d8ef` 就行了：

```
$ git reset e12d8ef --hard  //--hard 可以強迫放棄 Reset 之後修改的檔案
```

如果一開始沒有記下來 Commit 的 SHA-1 值也沒關係，當 *HEAD* 有移動的時候（例如切換分支或 reset 都會造成 HEAD 移動），Git 就會在 Reflog 裡記上一筆。

```
$ git reflog
657fce7 (HEAD -> master) HEAD@{0}: reset: moving to HEAD~2
e12d8ef (origin/master, origin/HEAD, cat) HEAD@{1}: checkout: moving
from cat to master
e12d8ef (origin/master, origin/HEAD, cat) HEAD@{2}: checkout: moving
from master to cat
```

. . .

[冷知識] HEAD 是什麼？

```
$ cat .git/HEAD
ref: refs/heads/master
```

. . .

六、使用分支

> *所謂的分支就只是一個有40 個字元的檔案而已，而這個分支，也就是這40 個字元，會標記出它目前指向哪一個Commit。*

[狀況] 不小心把還沒合併的分支砍掉了，救得回來嗎？

```
$ git branch -D cat
Deleted branch cat (was b174a5a).
```

我們在 master 砍掉 cat 這個 branch 後，東西還會存在。

> 分支只是一個指向某個 Commit 的指標，刪除這個指標並不會造成那些 Commit 消失。

刪掉分支，那些 Commit 還是在，只是因為你可能不知道或沒記下那些 Commit 的 SHA-1 值，所以不容易再拿來利用。

```
$ git branch new_cat b174a5a //建立一個 new_cat 指向這個commit
```

沒記下剛剛刪掉 cat 分支的 SHA-1 ？

從 `git reflog` 指令去翻翻看，Reflog 預設會保留 30 天，所以 30 天內應該都還找得到。

. . .

另一種合併方式（使用 rebase）

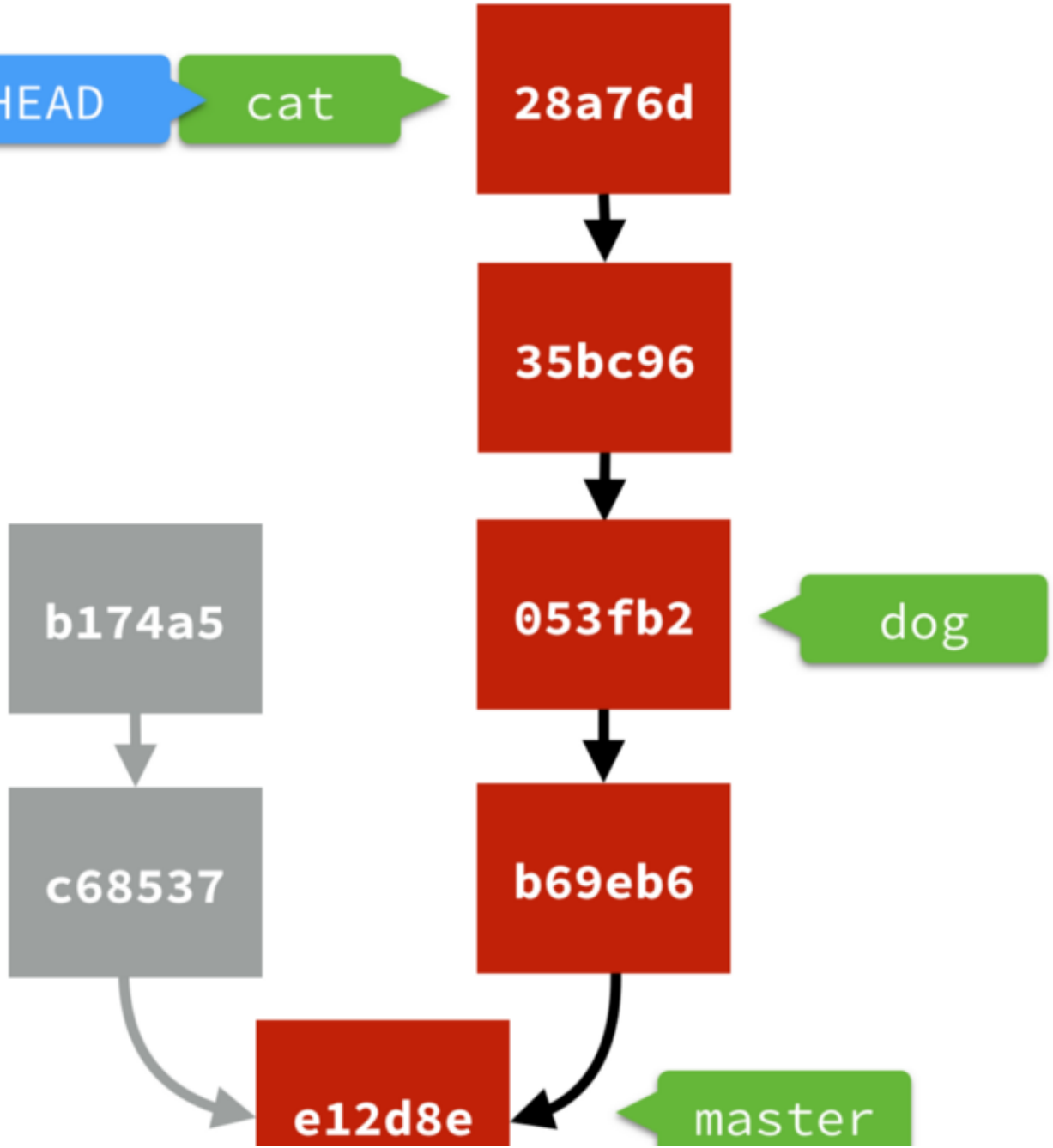
> *`git rebase` ，也可以用來做跟 `git merge` 類似的事情。*

從字面上看，「rebase」是「re」加上「base」，翻成中文大概是「重新定義分支的參考基準」的意思。「base」就是指「你這分支是從哪裡生出來的」。

假設目前有master、cat 以及 dog 三個分支，目前在 cat 分支上。這時：

```
$ git rebase dog
First, rewinding head to replay your work on top of it...
```

這指令大概就是「我，`cat` 分支，現在要重新定義我的參考基準，並且將使用 `dog` 分支當做我新的參考基準」的意思。
圖示：



rebase-branch

1. 「先拿 `c68537` 這個 Commit 接到 `053fb2` 這個 Commit 上」，因為 `c68537` 原本的上一層 Commit 是 `e12d8e` ，現在要接到 `053fb2` 上，所以需要重新計算這個 Commit 的 SHA-1 值，重新做出一顆新的 Commit 物件 `35bc96` 。
2. 「再拿 `b174a5` 這個 Commit 接到剛剛那個新做出來的 Commit 物件 `35bc96` 上」，同理，因為 `b174a5` 這顆 Commit 要接到新的 Commit 的原因，所以它也會重新計算 SHA-1 值，得到一個新的 Commit 物件 `28a76d` 。
3. 最後，原本的 `cat` 是指向 `b174a5` 這個 Commit，現在要改指向最後做出來的那顆新的 Commit 物件 `28a76d` 。
4. HEAD 還是繼續指向 `cat` 分支。

. . .

[狀況] 取消 rebase？

從上面看如果單純使用 `git reset HEAD^ --hard` 會回到上一個 commit，故不適用。

使用 ORIG_HEAD

這個 ORIG_HEAD 會記錄「危險操作」之前 HEAD 的位置。例如分支合併或是 Reset 之類的都算是所謂的「危險操作」。

```
git reset ORIG_HEAD --hard
```

. . .

Reset、Revert 跟 Rebase 指令有什麼差別？

Reset : 把目前的狀態設定成某個指定的 Commit 的狀態，通常適用於尚未推出去的 Commit。
Rebase : 不管是新增、修改、刪除 Commit 都相當方便，用來整理、編輯還沒有推出去的 Commit 相當方便，但通常也只適用於尚未推出去的 Commit。
Revert : 新增一個 Commit 來反轉（或說取消）另一個 Commit 的內容，原本的 Commit 依舊還是會保留在歷史紀錄中。雖然會因此而增加 Commit 數，但通常比較適用於已經推出去的 Commit，

2018 / 7 /25 更新

如何修改已 push 的 commit？

修改完專案後推上去才發現之前有個多餘的 Commit 需要拿掉，如果還是自己的 remote 的話，雖然不建議，但可用強推的方式蓋掉。
先 reset HEAD 到那個版本前，修改完後 add , commit 後：

```
git push -f [遠端名稱] [branch名稱]
```

這樣即可修改已推出去的 Commit 。
（完）

Git Learning Notes

WRITTEN BY

Leo Kao

Follow

資工人. 背包客. 攝影手。對新事物充滿好奇心、談論天馬行空。喜好人與人之間的互動，期望以科技帶給人際間更友善的服務。https://www.linkedin.com/in/chia-hung-kao-03641ab0/ Mail: leokao0726@gmail.com

See responses (1)

More From Medium

Related reads

How To Write an iOS Developer Resume That Will Land You an Interview

Gareth Cheng in Better...
Jul 7 · 6 min read ★

585

Related reads

It is still Day 1 in the age of convenience

Cristina Berta Jones
Jun 7, 2018 · 7 min read

17

Related reads

Humans Of The Web

Abhishek Kothari
Sep 26, 2018 · 5 min read ★

51