

# 寫給大家的 *Git* 教學

*Littlebtc (Hsiao-Ting Yu)*

本簡報內容基於Scott Chacon的「Pro Git」電子書，修改過後  
的內容以CC-BY-NC-SA-3.0授權發佈





*What is Version Control?*  
*What is Git?*



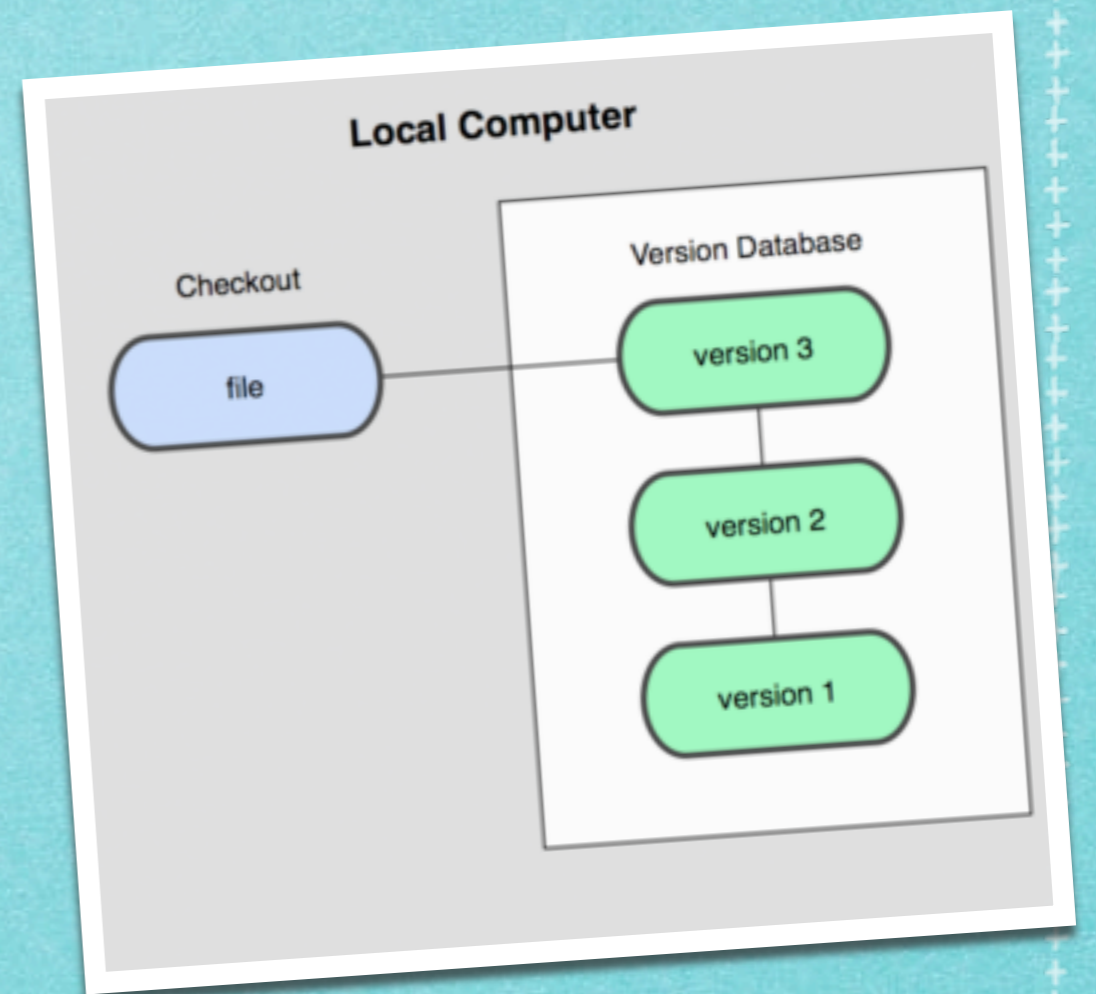
# 為什麼要版本控制？

- ▶ 「凡走過必留下痕跡」
  - ▶ 改了東西，不會改不回來
- ▶ 「三個臭皮匠勝過一個諸葛亮」
  - ▶ 大家一起改，還能清楚知道對方改了什麼



# Local 式版本控制

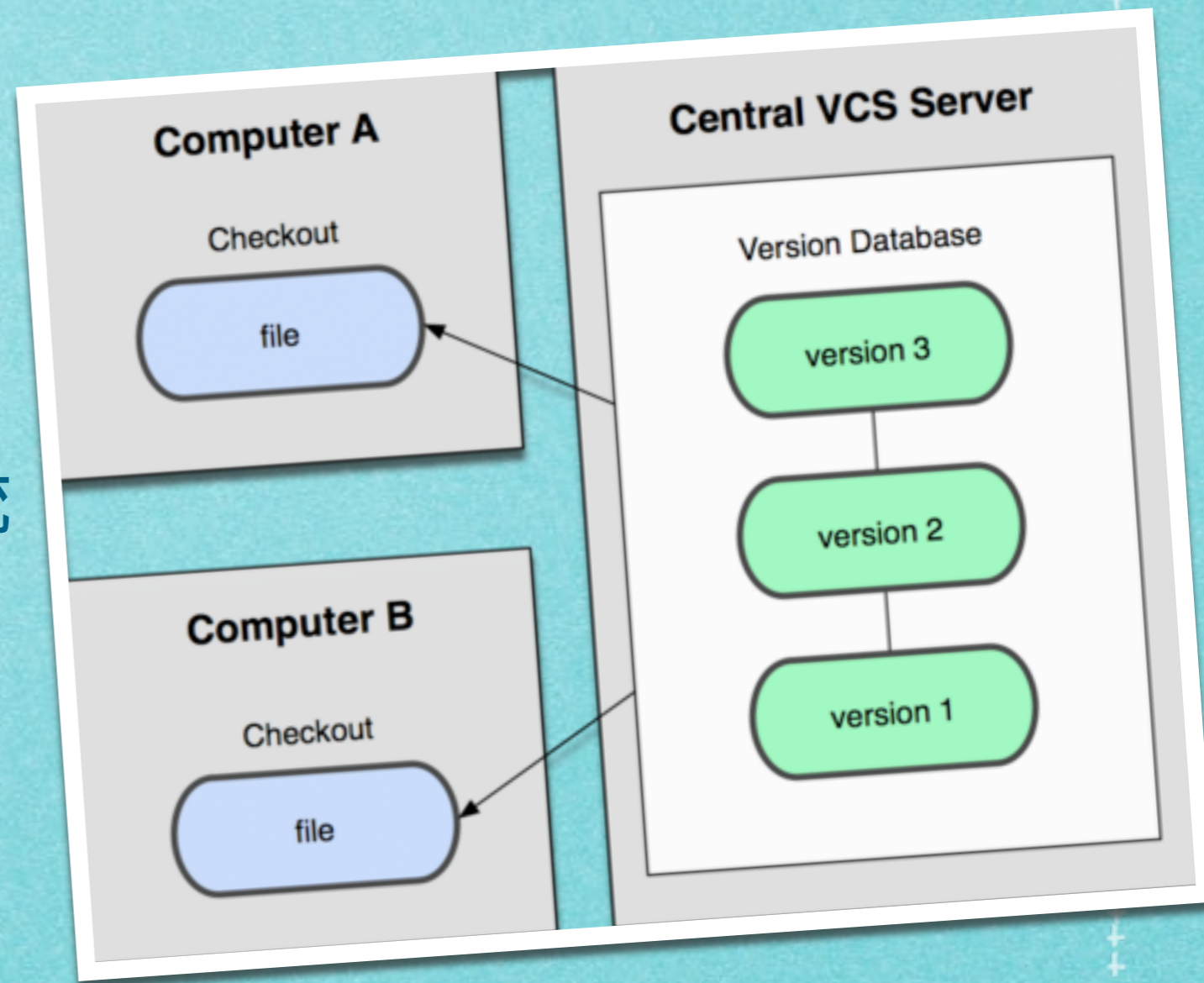
- ▶ 在「自己的電腦裡」  
建立一個版本資料庫
- ▶ rcs 指令
- ▶ 問題：如何協作？





# 中心式版本控制

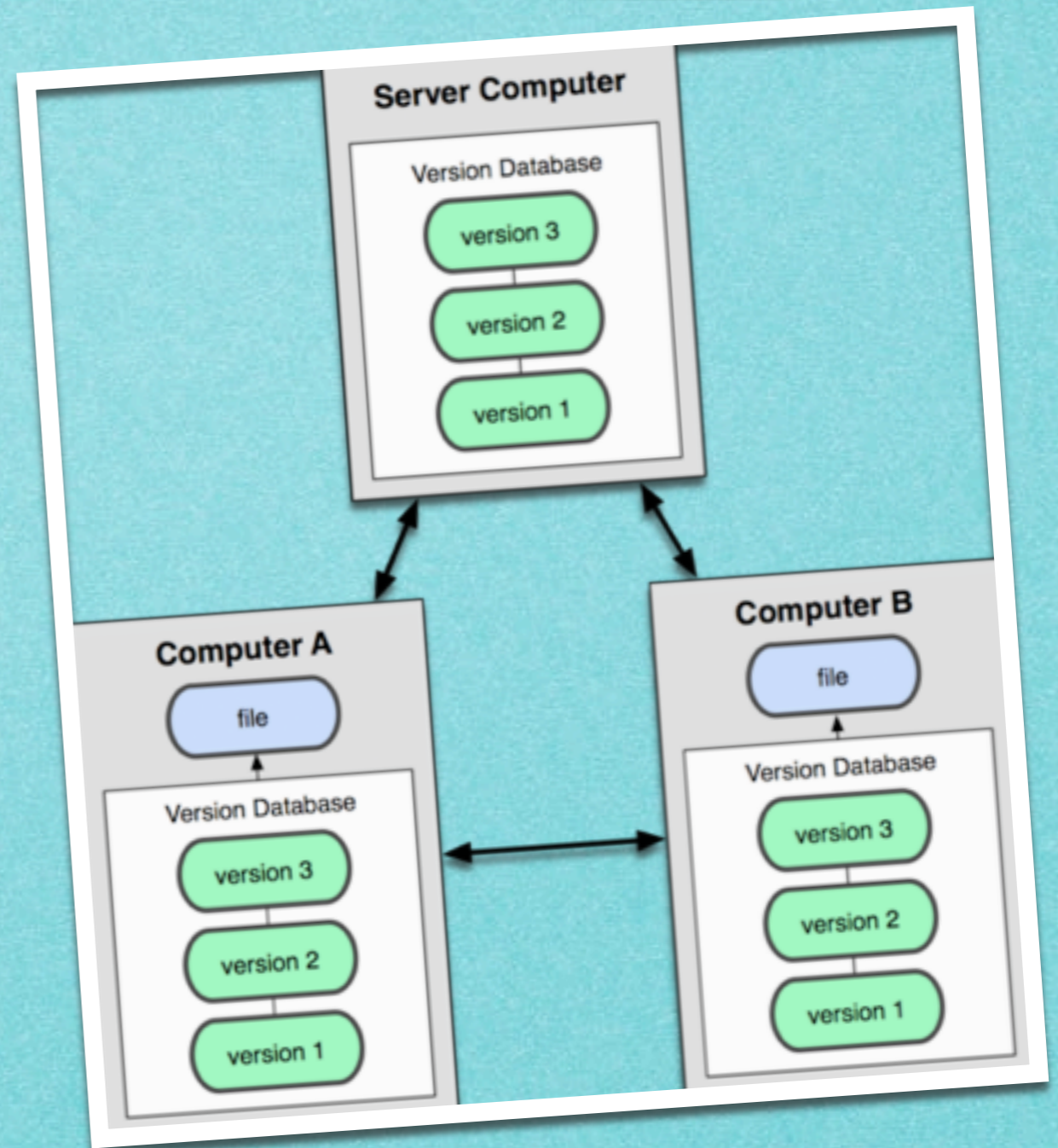
- ▶ 在一台Server上，儲存所有的版本記錄
- ▶ 從裡頭Checkout  
將變更Commit上去
- ▶ Subversion (SVN) 是主流
- ▶ 問題：
  - ▶ Server會掛！
  - ▶ 開發太亂！





# 分散式版本控制

- ▶ 「大家都有完整資料庫」
- ▶ 「大家都能獨立工作」
- ▶ 「Server爛了不要緊，拿到一份好的資料庫灌回去就可以全部復原！」
- ▶ 現在爆紅的 `git`, `Mercurial` (`hg`), `bazaar` (`bzr`) 都屬此類
- ▶ 問題：亂成一團???
- ▶ 用好的Branch機制來解決！





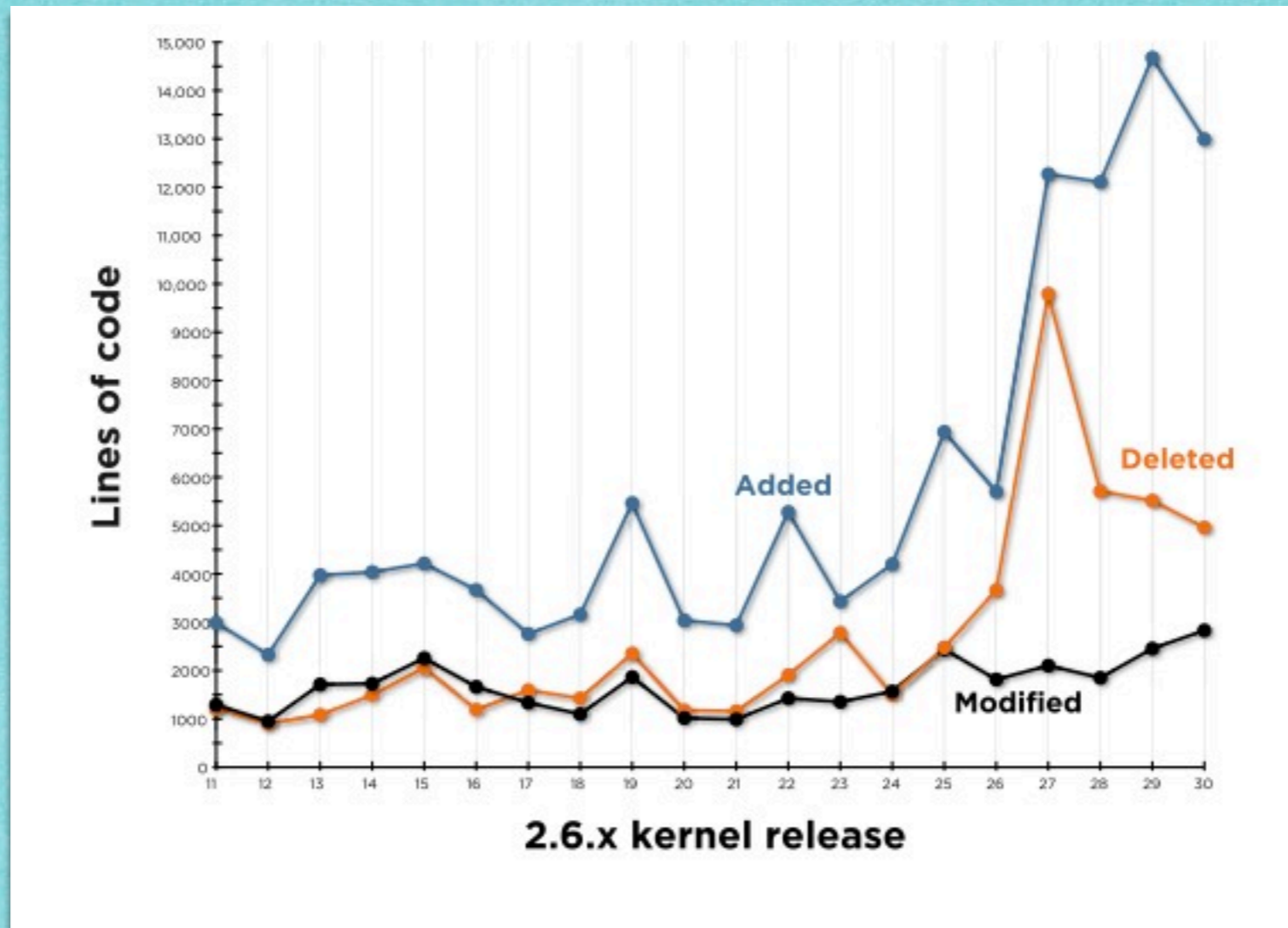


# 從 *Git* 的誕生講起

*Linux Kernel* 是一切的開端

([http://en.wikipedia.org/wiki/Linux\\_kernel](http://en.wikipedia.org/wiki/Linux_kernel), Wikipedia contributors, CC-BY-SA)





# 極端龐大的資訊量

( From: *Linux Kernel Development: How Fast it is Going, Who is Doing It, What They are Doing, and Who is Sponsoring It, Fair use(?)* )



# 先前的嘗試：BitKeeper

- ▶ Linus Torvalds：「我就是不喜歡中央式的系統！」
  - ▶ 所以 Linux Kernel 以前的變更都是用 Patch + 壓縮檔來散布
- ▶ 2002年，Linus 採用 BitKeeper 分散版本控制系統（專有！）
- ▶ 社群不滿，甚至嘗試逆向工程，於是跟 BitKeeper 開發商鬧翻
- ▶ Git 因此而生
- ▶ (pic: [http://commons.wikimedia.org/wiki/File:Linus\\_Torvalds.jpeg](http://commons.wikimedia.org/wiki/File:Linus_Torvalds.jpeg) CC-BY-SA-3.0 & GFDL)





# Git 的設計目標

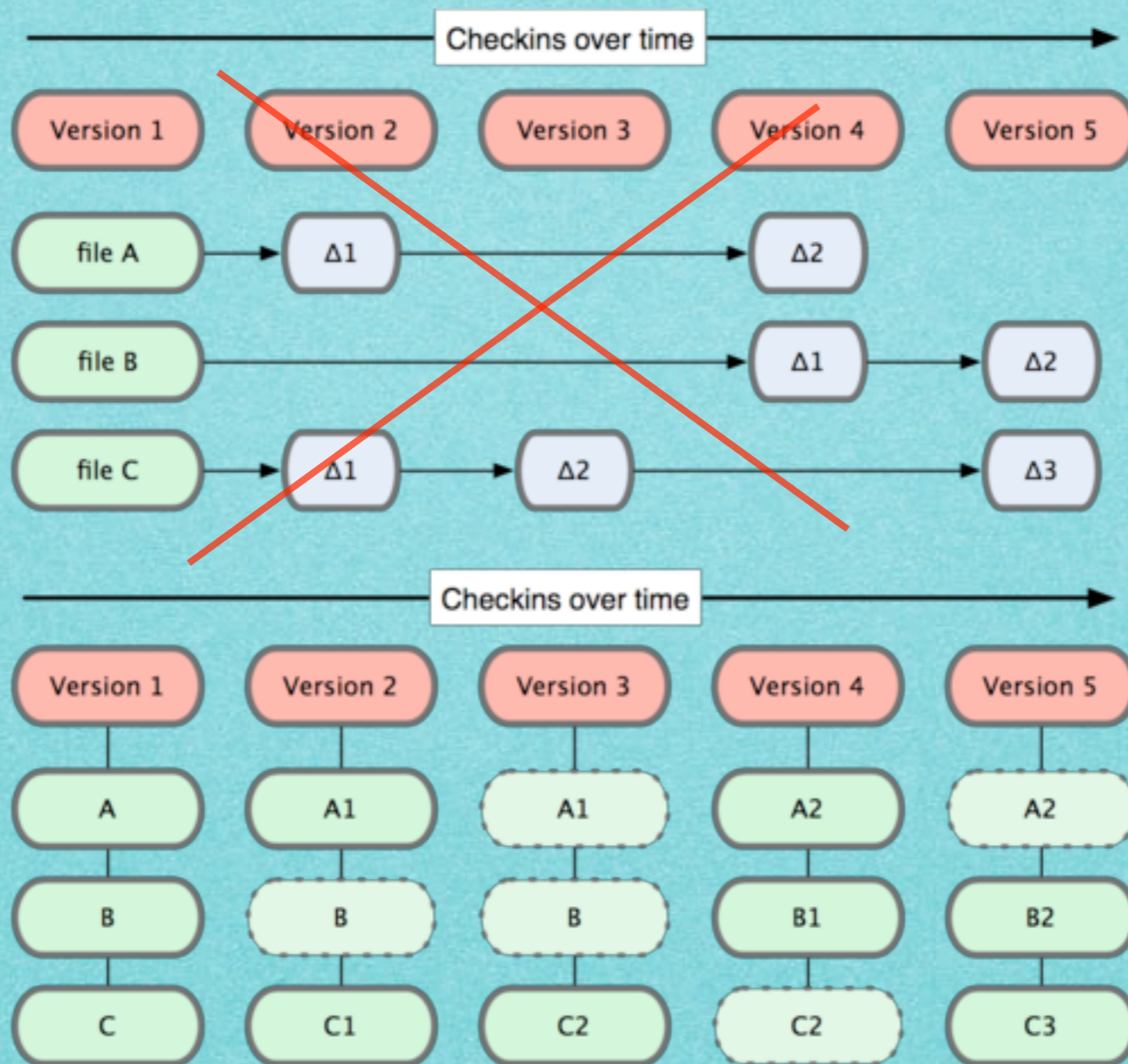
- ▶ 超快
- ▶ 超簡單
- ▶ 支援非線性的開發
  - ▶ 大家都可以自己改，改完合併來合併去，不必受到一條主線的拘束
- ▶ 完全分散式
- ▶ 可以處理 Linux Kernel 「超大」的資料量
- ▶ And then realize that nothing is perfect. Git is just *\*closer\** to perfect than any other SCM out there. -- Linus on Git mailing list



# *Git Basics*



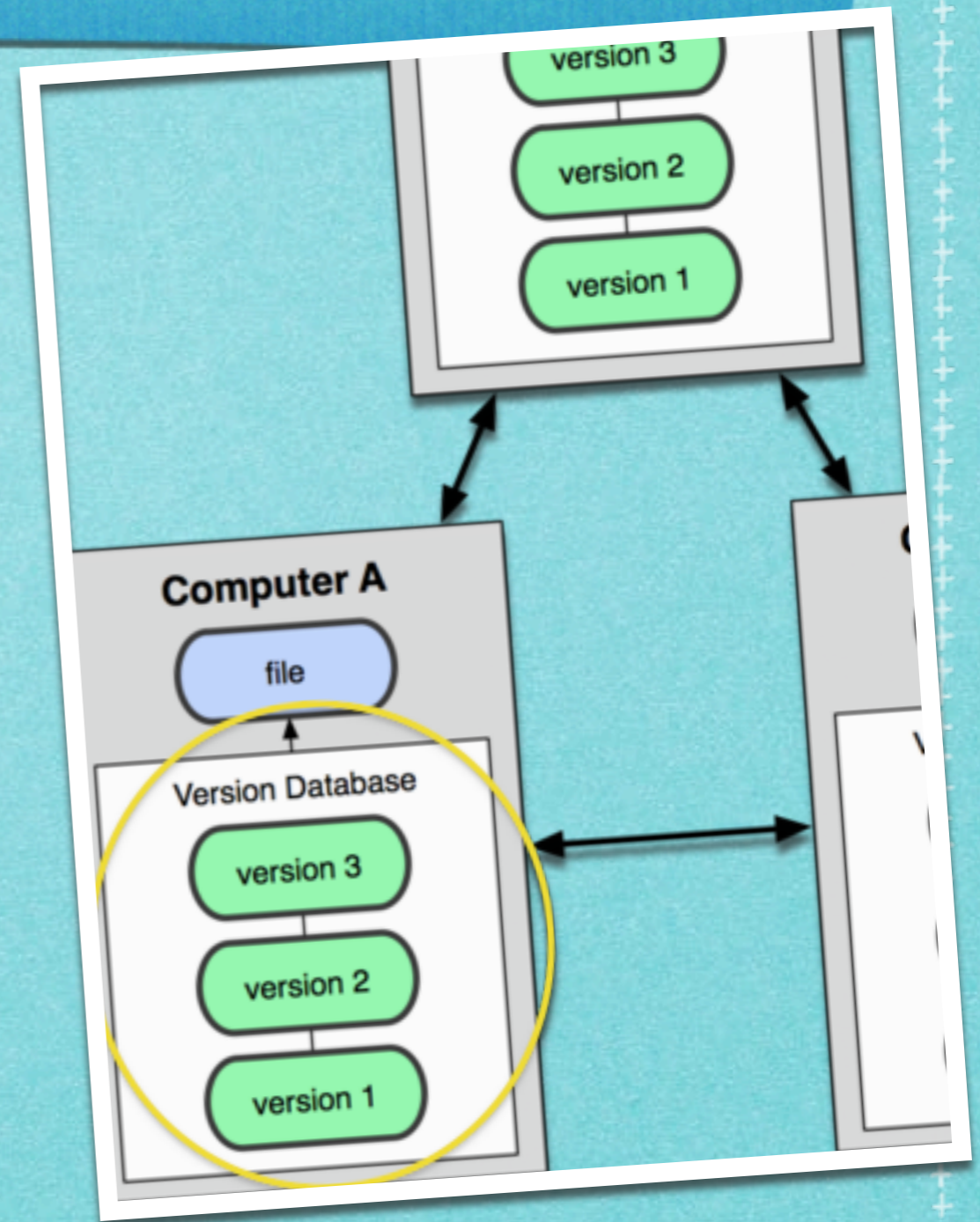
# Git不是記「檔案的差異」， 而是記錄「檔案的快照」





# Git 幾乎什麼事都在本機進行！

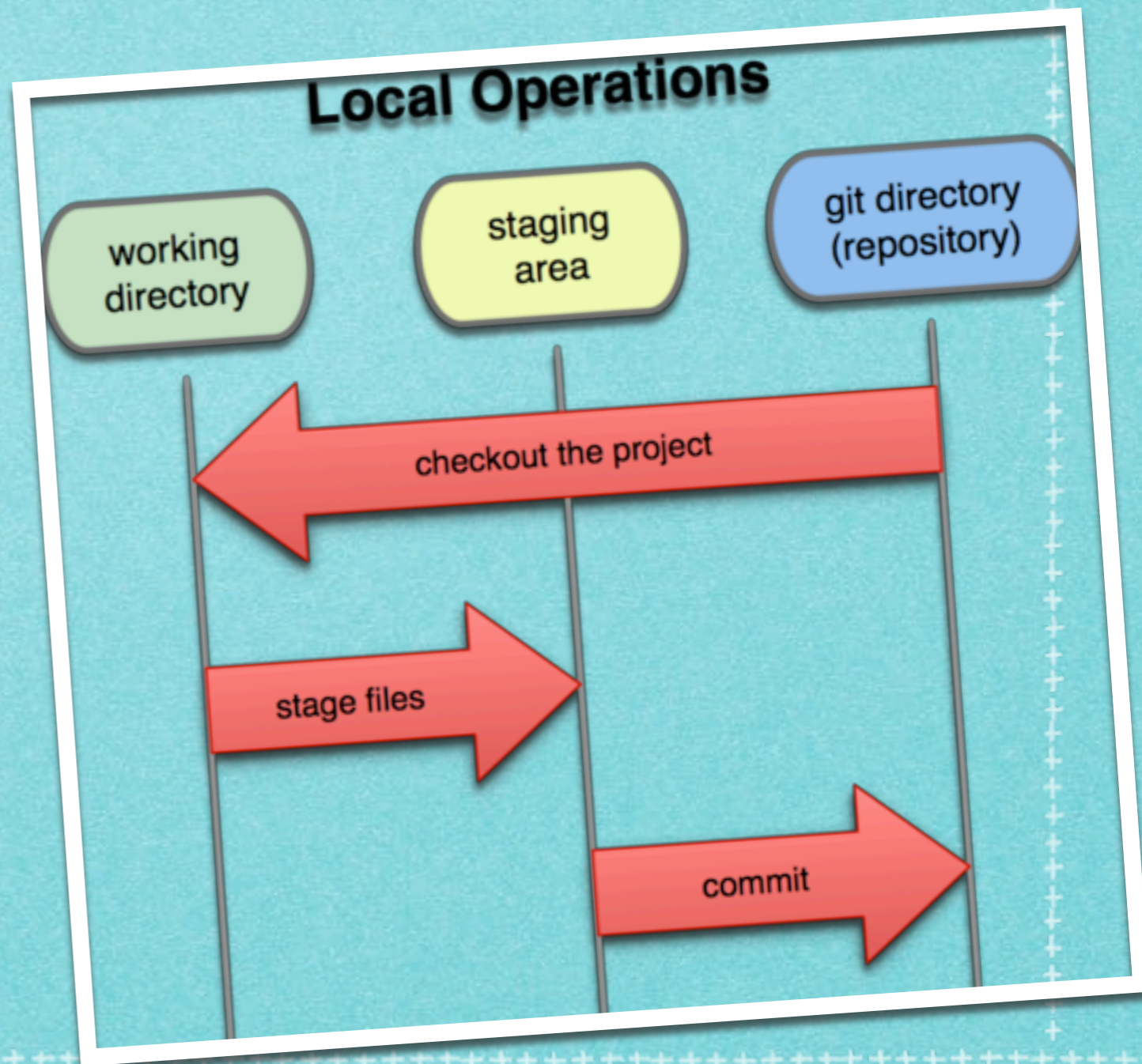
- ▶ 你不只有一份完整的資料庫.....
- ▶ 閱讀版本歷史、提交變更這些動作都可以在本機進行
- ▶ 不需要網路連線也能單獨工作！





# Git 的檔案處理

- ▶ 使用 Checksum 來確保檔案的完整性
- ▶ 設計上「只會增加資料」，因此東西可以輕鬆復原
- ▶ 在本機的檔案管理中，增添了「Staging」的觀念





# *Using Git*



# 建立一個新的 Git Repository

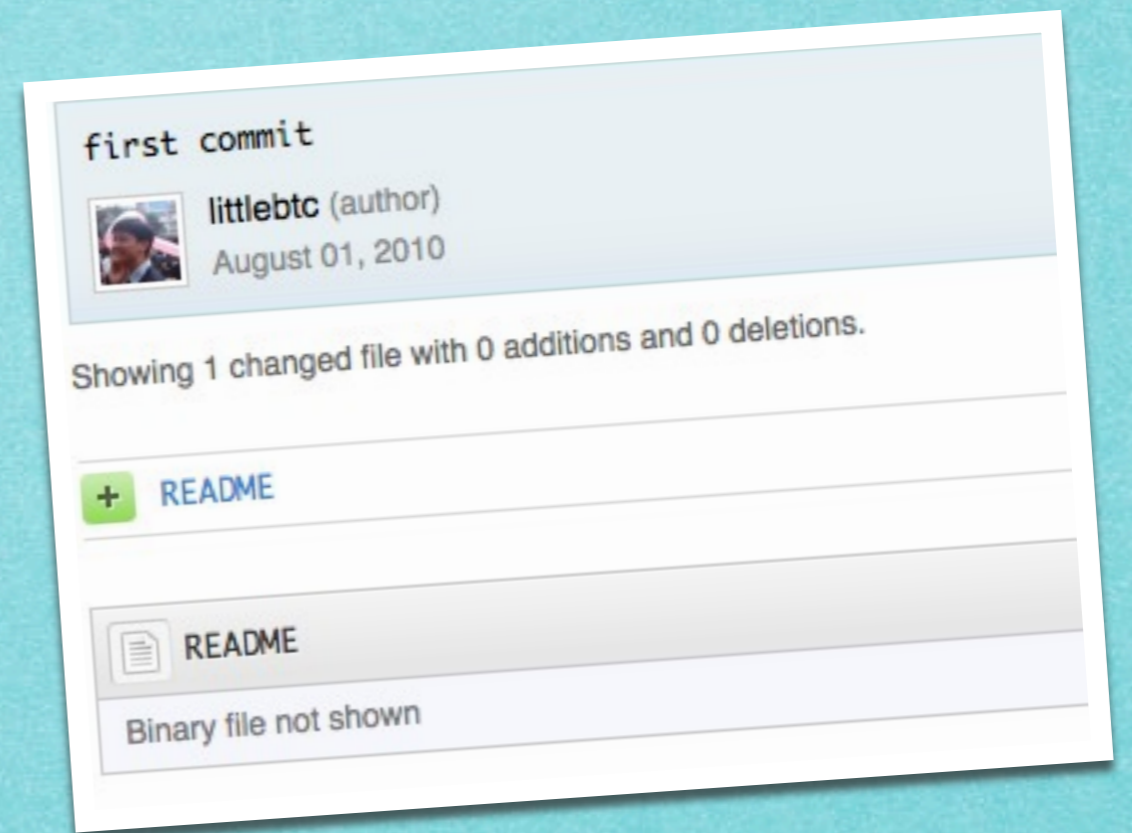
- ▶ 「Repository」（倉庫、套件庫）就是一份版本控制中的「版本資料庫」
- ▶ 找一個空的資料夾，然後執行：  
`git init`
- ▶ 套件庫需要的資訊會放在.git資料夾裡



# Initial Commit

- ▶ 新增一個檔案README
- ▶ 把所有檔案加進去
- ▶ 然後提交變更
- ▶ 

```
touch README  
git add .  
git commit -m 'Initial commit'
```





# 複製別人的 Repository

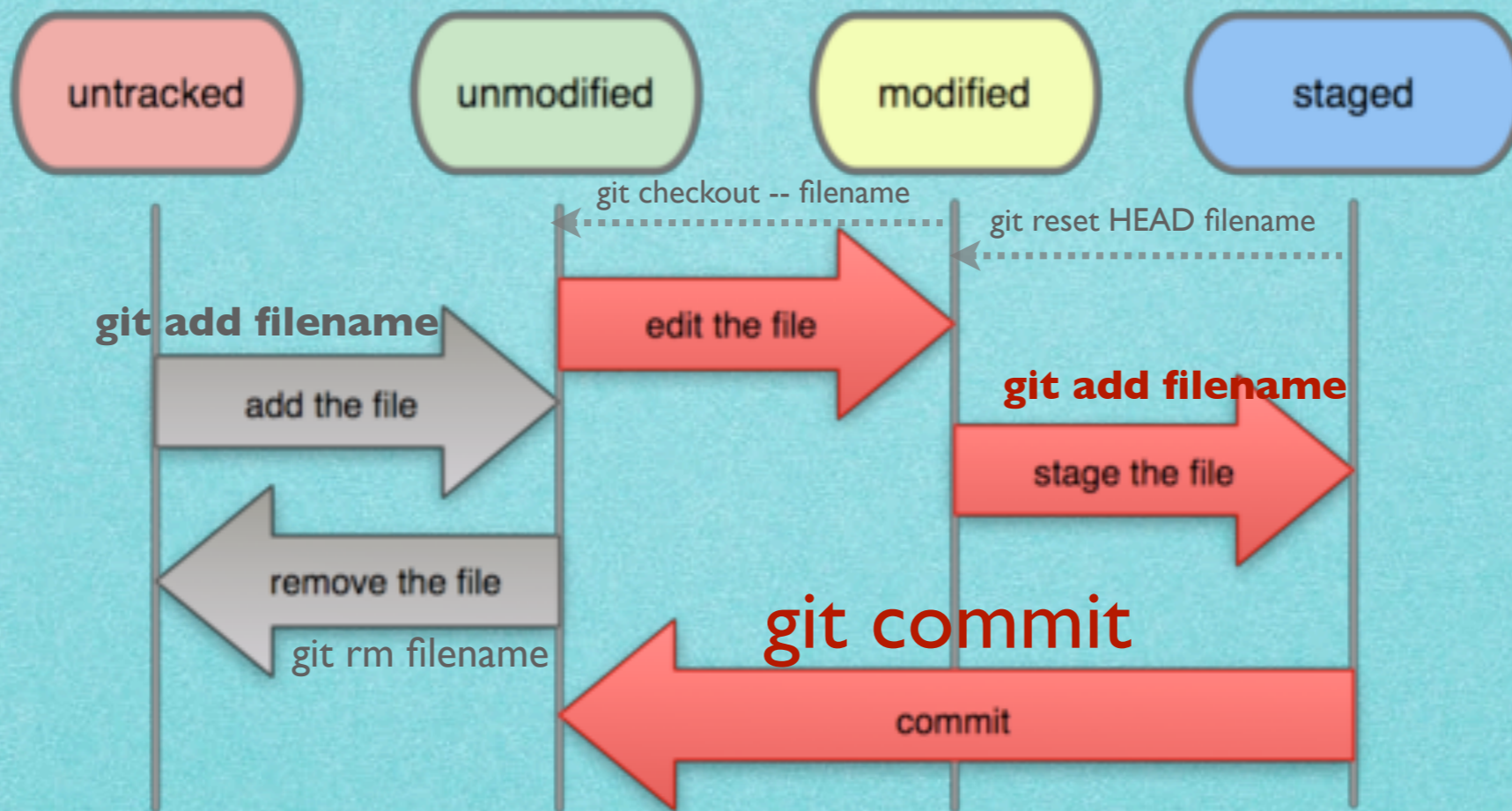
Clone的專案會放在目前路徑下的新目錄。

```
$ git clone http://github.com/jquery/jquery.git  
Initialized empty Git repository in /Users/littlebtc/work/jquery/.git/  
remote: Counting objects: 14043, done.remote: Compressing objects: 100%  
(4070/4070), done.  
remote: Total 14043 (delta 9717), reused 13753 (delta 9452)  
Receiving objects: 100% (14043/14043), 12.45 MiB | 288 KiB/s, done.  
Resolving deltas: 100% (9717/9717), done.
```



# File Status

## File Status Lifecycle



- 在 `.gitignore` 檔案中列出的檔案名稱將被忽略  
(`.gitignore` 這個檔也要 `commit`!)
- 使用 `git status` 來檢查檔案狀態



# git status

```
littlebtc-MBP:ccmediacollector littlebtc$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   install.rdf
#
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   content/collection/collection.js
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       modules/ContentFetcher.jsm
#       skin/note.png
```



## 如何檢視Commit log ?

- git log (console)
- GitX (Mac)
- giggle (Linux) (截圖來自官網)

```
commit 55bfb94547350744f029a3cd4501e0c833a7c1d1
Author: Hsiao-Ting Yu <sst.dreams@gmail.com>
Date: Thu Oct 14 08:24:08 2010 +0800

Empty babelzilla-locale since there will have lots of changes on it

commit eea5dacfca545308b2826b78a4ae81a4feb54c30
Author: Hsiao-Ting Yu <sst.dreams@gmail.com>
Date: Wed Oct 13 22:16:45 2010 +0000

Compatibility -> 4.0b8pre

commit 7ed45a2865684b5d18de3133213a2cb1a7d90567
Author: Hsiao-Ting Yu <sst.dreams@gmail.com>
Date: Fri Oct 8 03:38:09 2010 +0000

combine smilefoxPanel / firefox_overlay

commit ae37a2928b9867d6ae74b6241e8acb1487deb208
Author: Hsiao-Ting Yu <sst.dreams@gmail.com>
Date: Fri Oct 8 03:37:53 2010 +0000

combine smilefoxPanel / firefox_overlay

commit f3fc9e4a8f56dc12e5a5efab2c372a35ac262db3
Author: Hsiao-Ting Yu <sst.dreams@gmail.com>
Date: Fri Oct 8 03:36:21 2010 +0000

Bug fixes

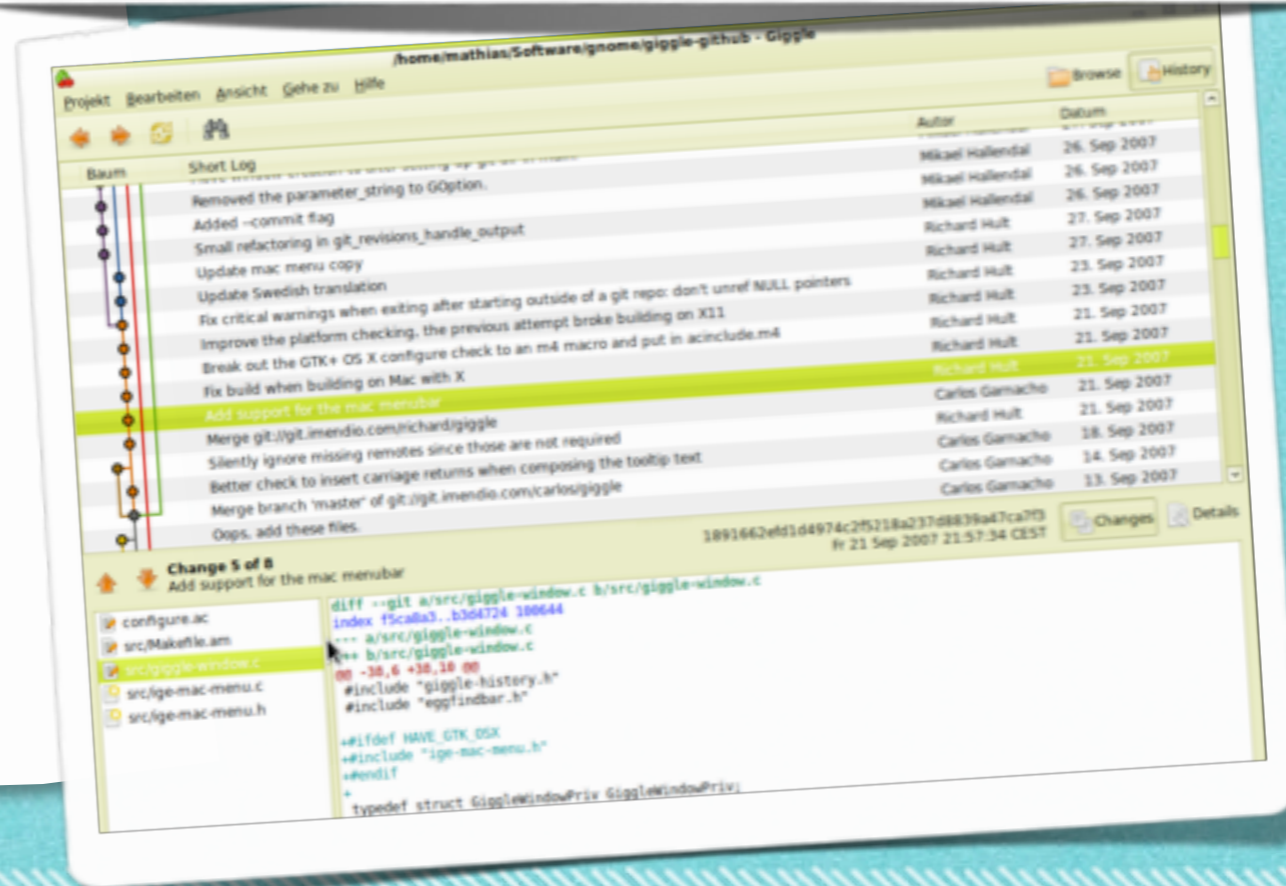
commit e8e385679cef48b83f1c17f1b1a180e475c7cf95
Author: Hsiao-Ting Yu <sst.dreams@gmail.com>
Date: Fri Oct 8 03:35:23 2010 +0000

Try to fix conflict problem for "early rejection".

commit 7ec150b22a646f1eef390858617bc7b851b9edf
Author: Hsiao-Ting Yu <sst.dreams@gmail.com>
Date: Wed Sep 15 00:25:16 2010 +0000

Compatibility: -> 4.0b7pre

commit a7781d2bfd84991dfc2e5e30372a0b0ab9b9b09
Author: Hsiao-Ting Yu <sst.dreams@gmail.com>
Date: Sat Sep 11 23:23:17 2010 +0000
```





# 改爛了...!?

- ▶ 改變最後一次的 Commit :  
`git commit --amend`
- ▶ 把 Stage 的檔案給 Unstage :  
`git reset HEAD file`
- ▶ 把修改過的檔案回到未修改狀態 :  
`git checkout -- file`



# Remote Server!!


- ▶ Branch 和 Remote 之間的互動！
  - ▶ 預設的 Branch 叫 master（稍後詳述）
  - ▶ 預設的 Remote 叫 origin
- ▶ `git pull origin =`  
`git fetch origin + git merge origin/master`  
拉下來之後 Merge
- ▶ `git push origin master`  
把 Master 給 Push 到 Origin



# "Remote" 的 Protocol

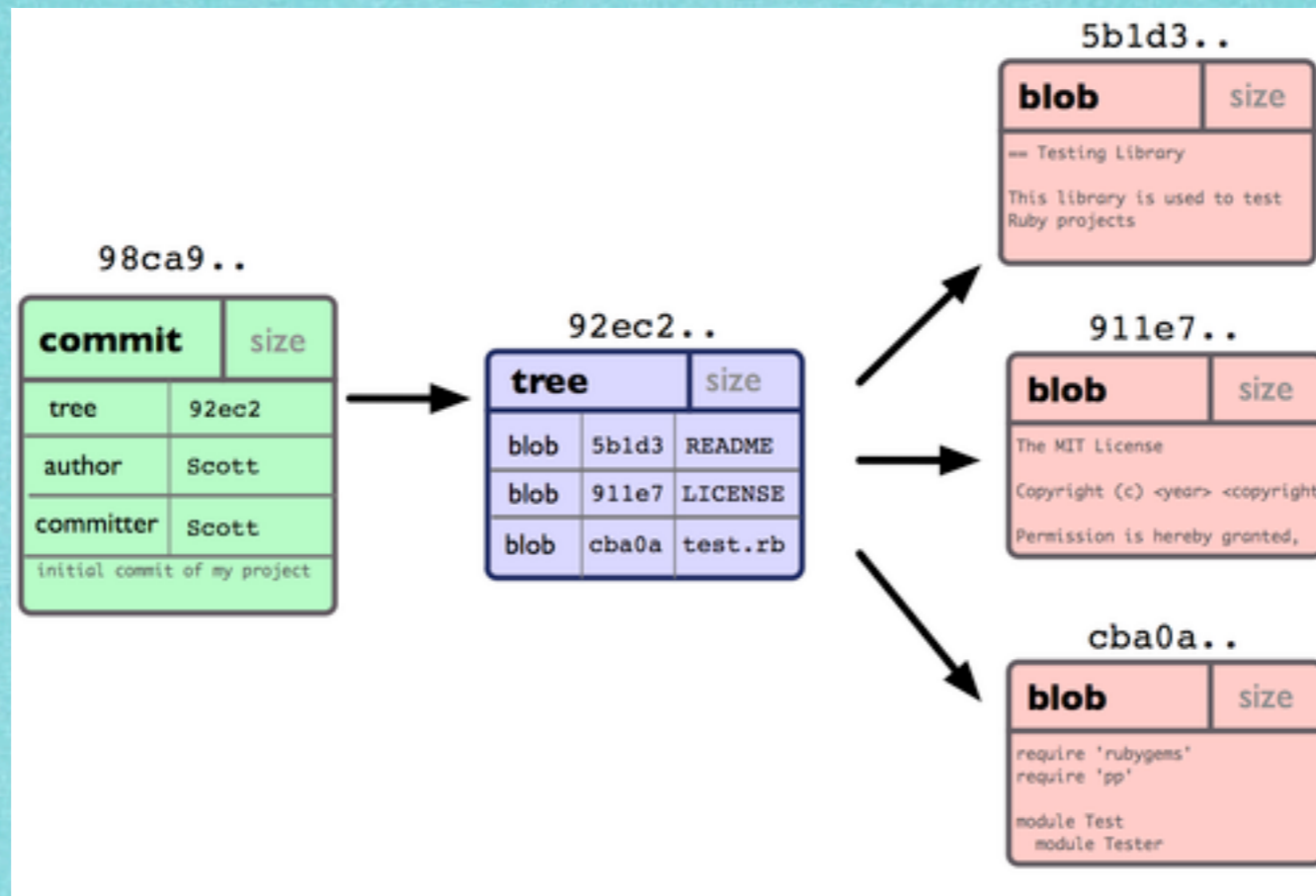
- ▶ **file://** (本機)
- ▶ **ssh://** (效率好、匿名訪問不可、適合寫入)
- ▶ **git://** (效率很棒、不好設定、適合讀取)
- ▶ **http:// or https://** (簡單、效率低)





*Branches*

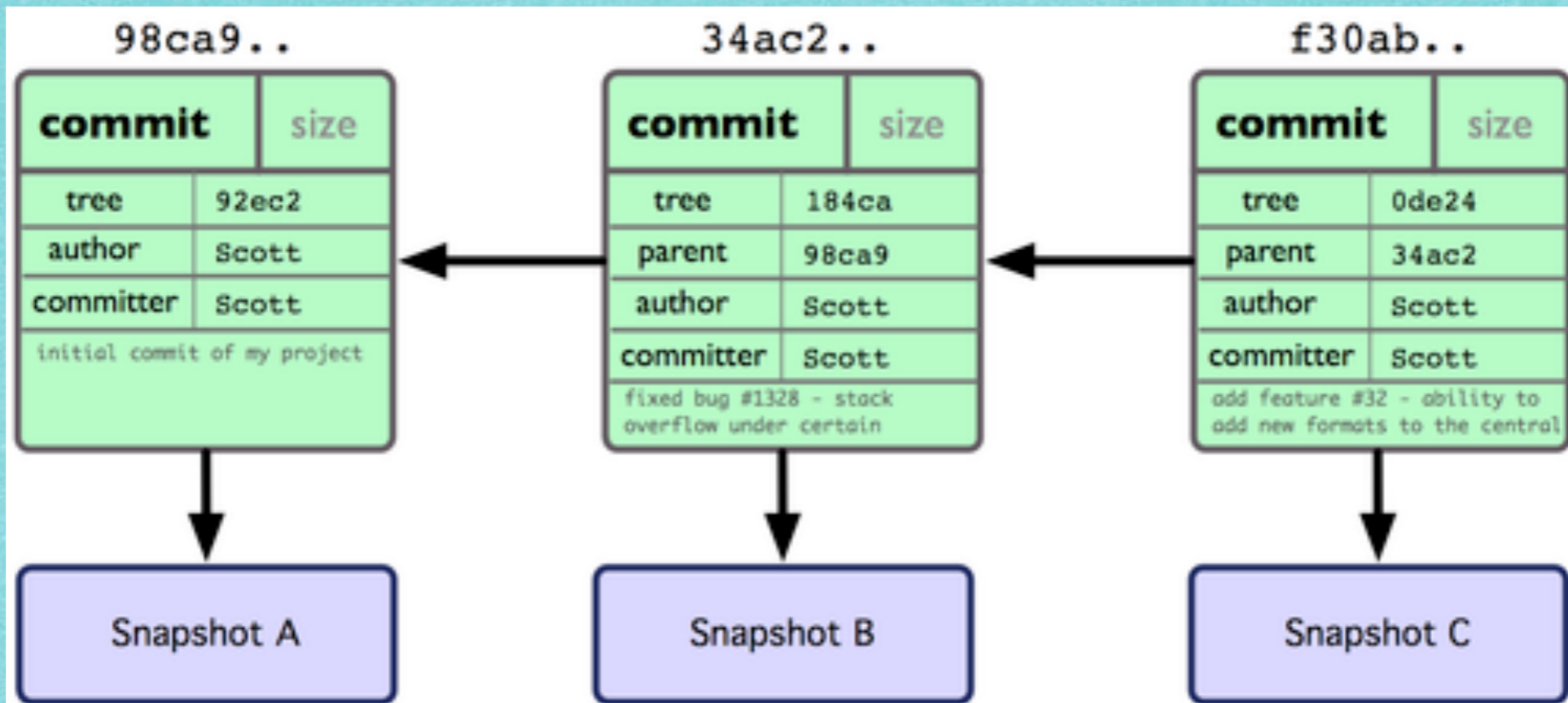




# 先從 *Commit* 說起

*git* 是這樣看待 *Commit* 的



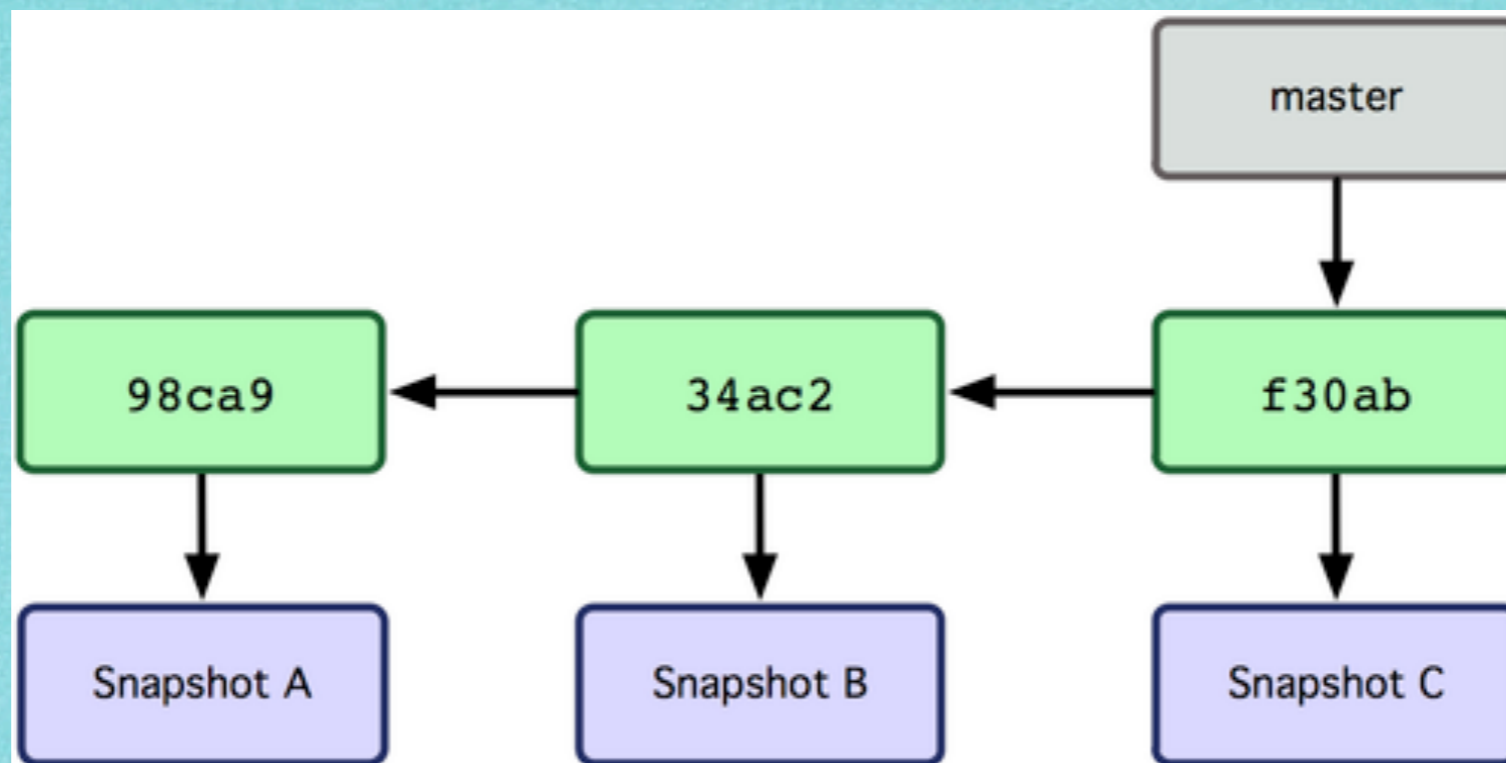


很多 *commit*

*Commit*之間互相連結

每個 *Commit* 對應一個 *Snapshot Tree*

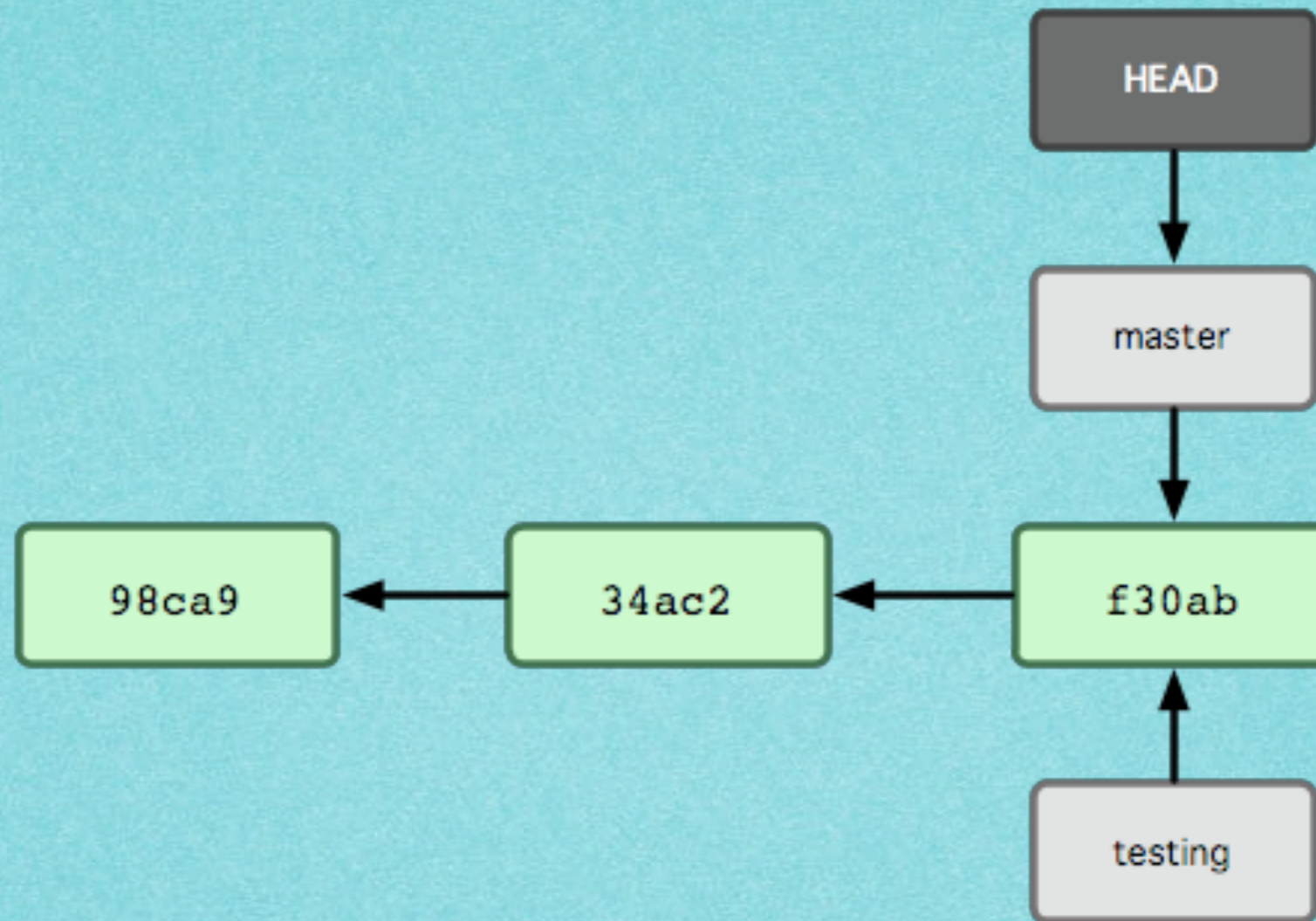




於是有了 *branch*

「標記我們在哪個 *commit* 上」

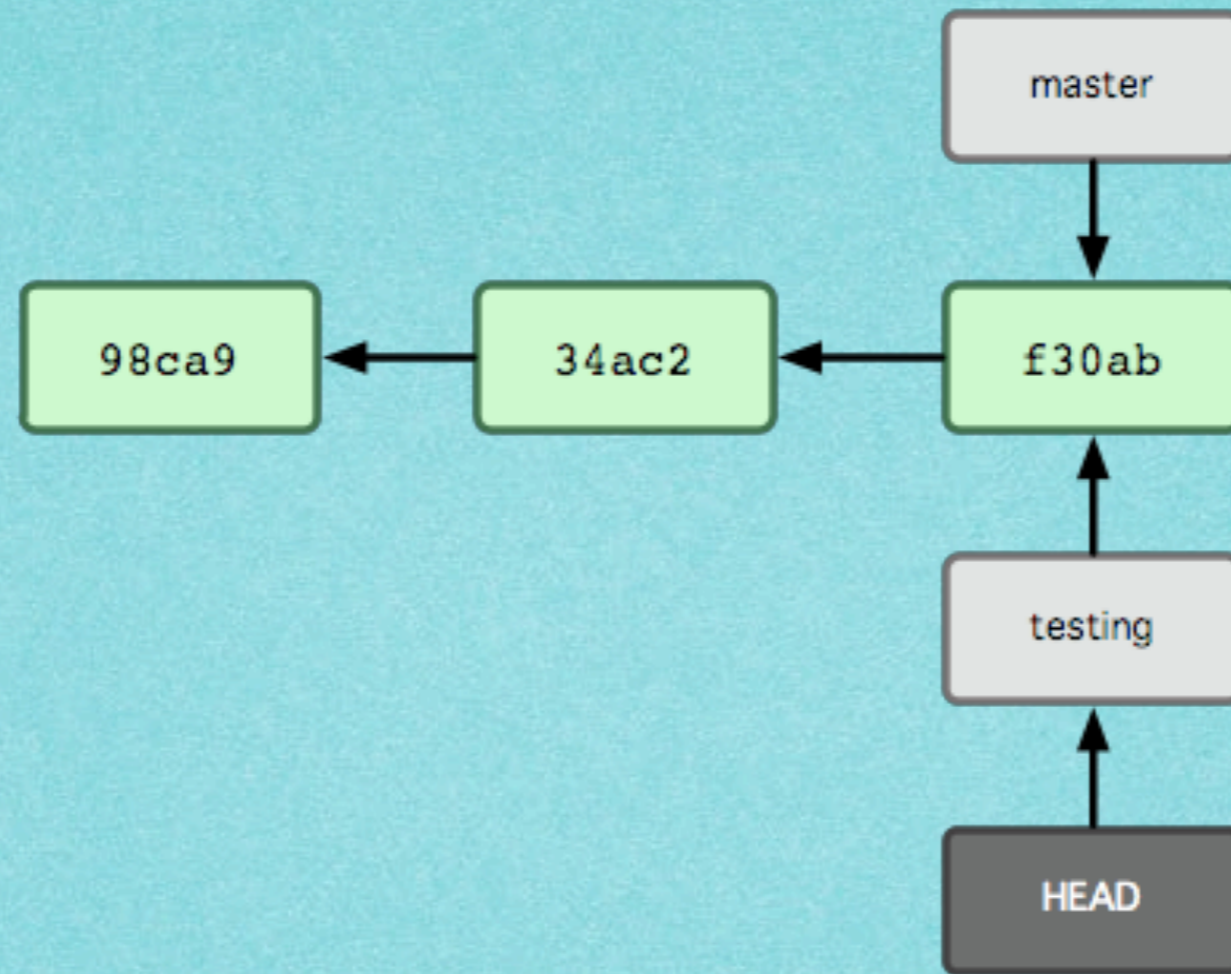




## git branch testing

新增一個"*testing*" branch  
*git*會使用「*HEAD*」紀錄目前所在的Branch



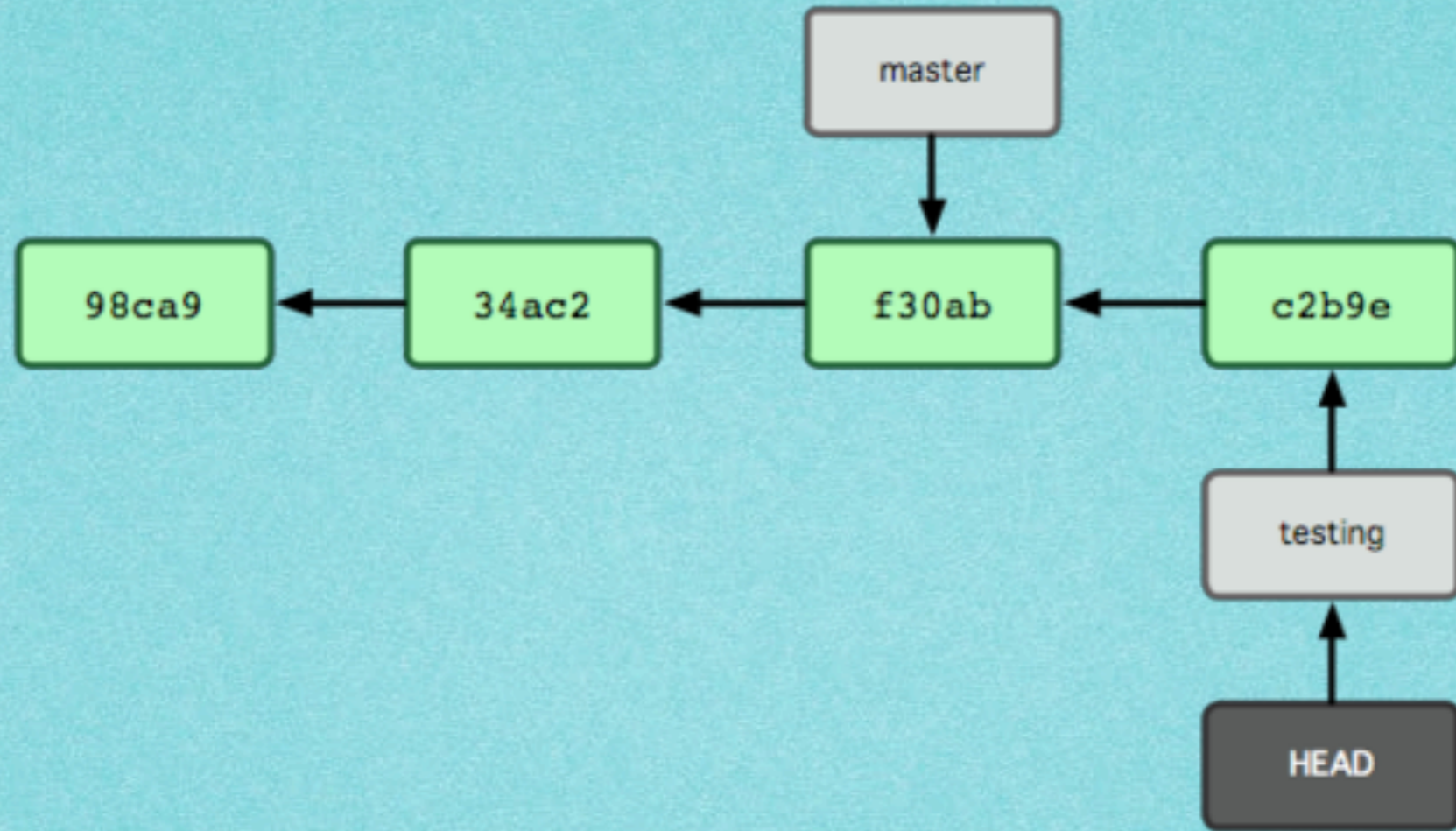


# git checkout testing

把「*HEAD*」變成 *Testing Branch*

or `git checkout -b testing` : 開新 *Branch* 過後切

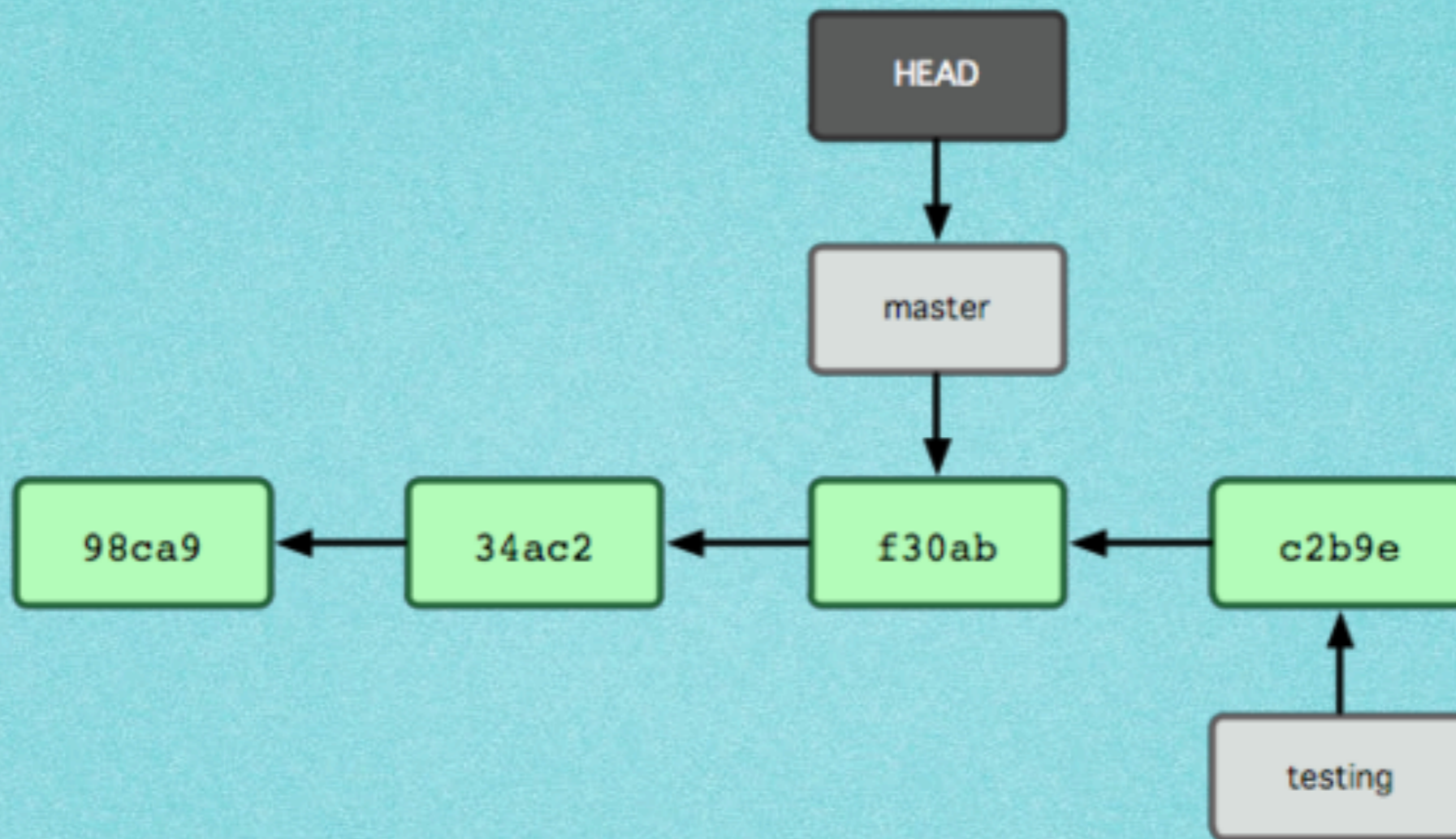




這時如果 *Commit* 下去...

變成只算在 *testing* 不算 *master*

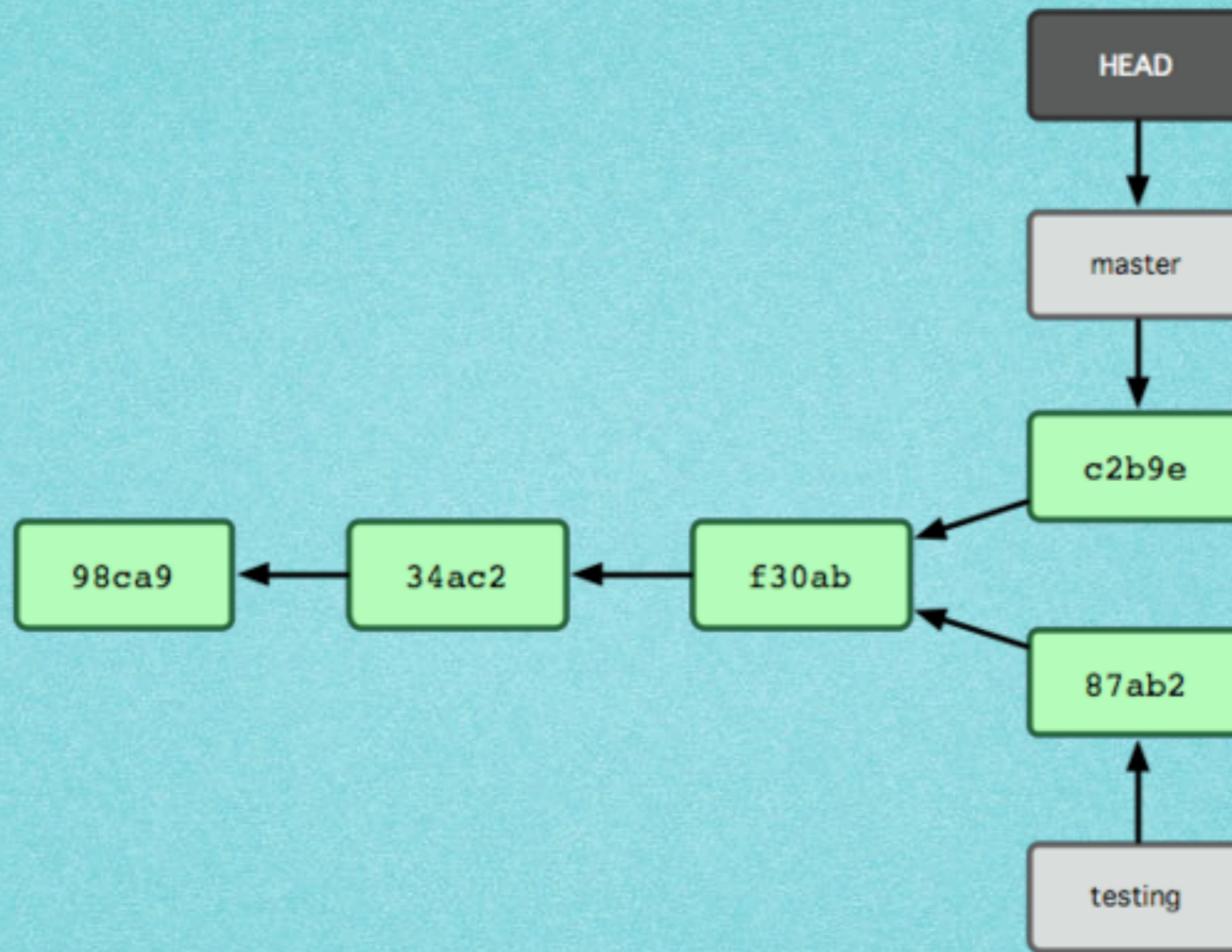




git checkout master

回到Master了，那麼What's next?





`git commit -m 'Change the master!'`

岔開來了！



# 如何重新變回一條？

## ▶ Merge

▶ 把岔開來的分支在往後「合起來」

▶ 通常的建議方式

## ▶ Rebase

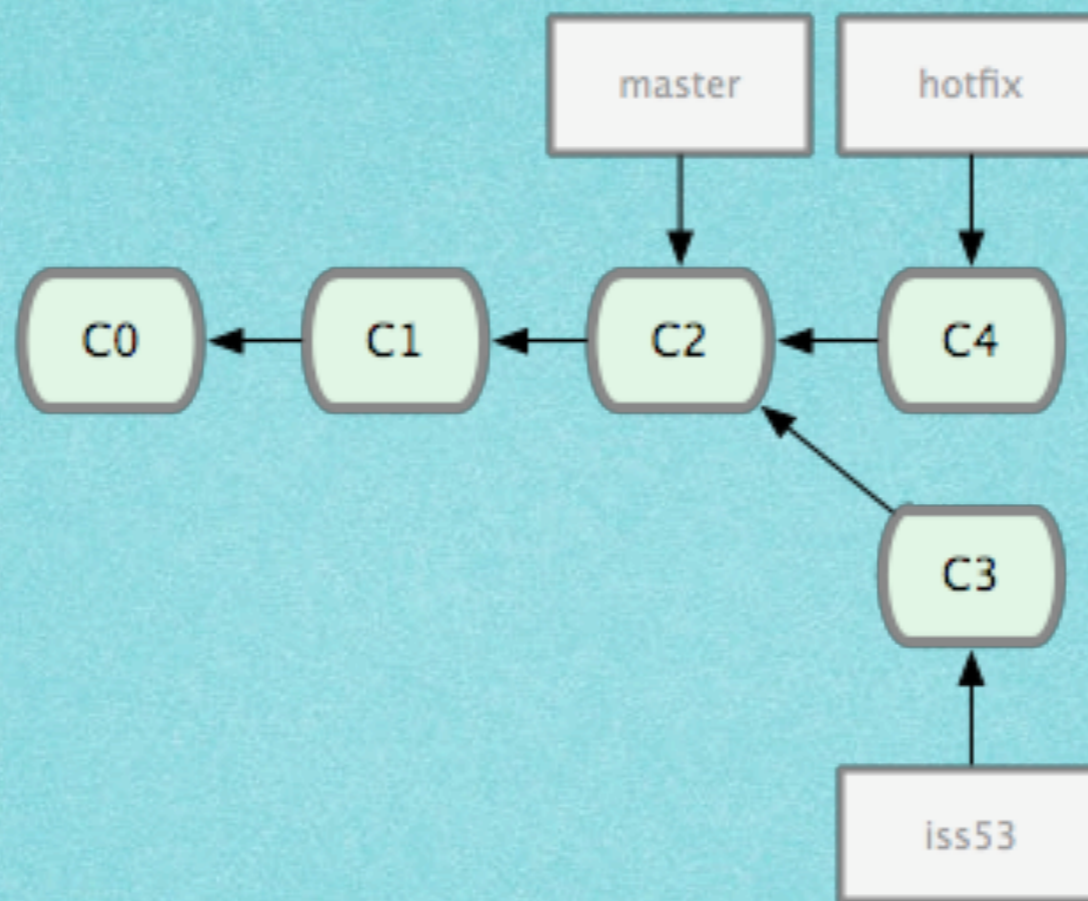
▶ 把岔開來的分支「裝回去」主線

▶ 「還沒Push出去的東西」才可以Rebase！



*Merge*

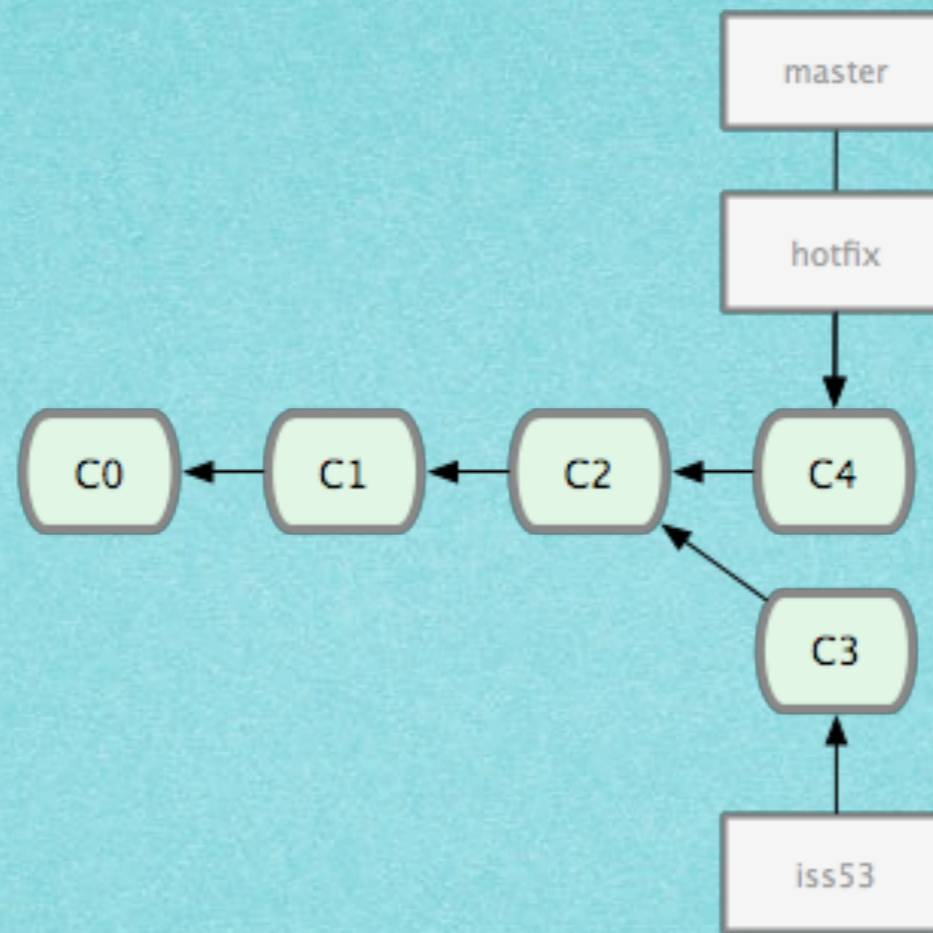




## Branch 滿天飛

有緊急的 *Bug*，我開了一個 *hotfix branch* 弄好了。  
現在要怎麼處理 *master*？

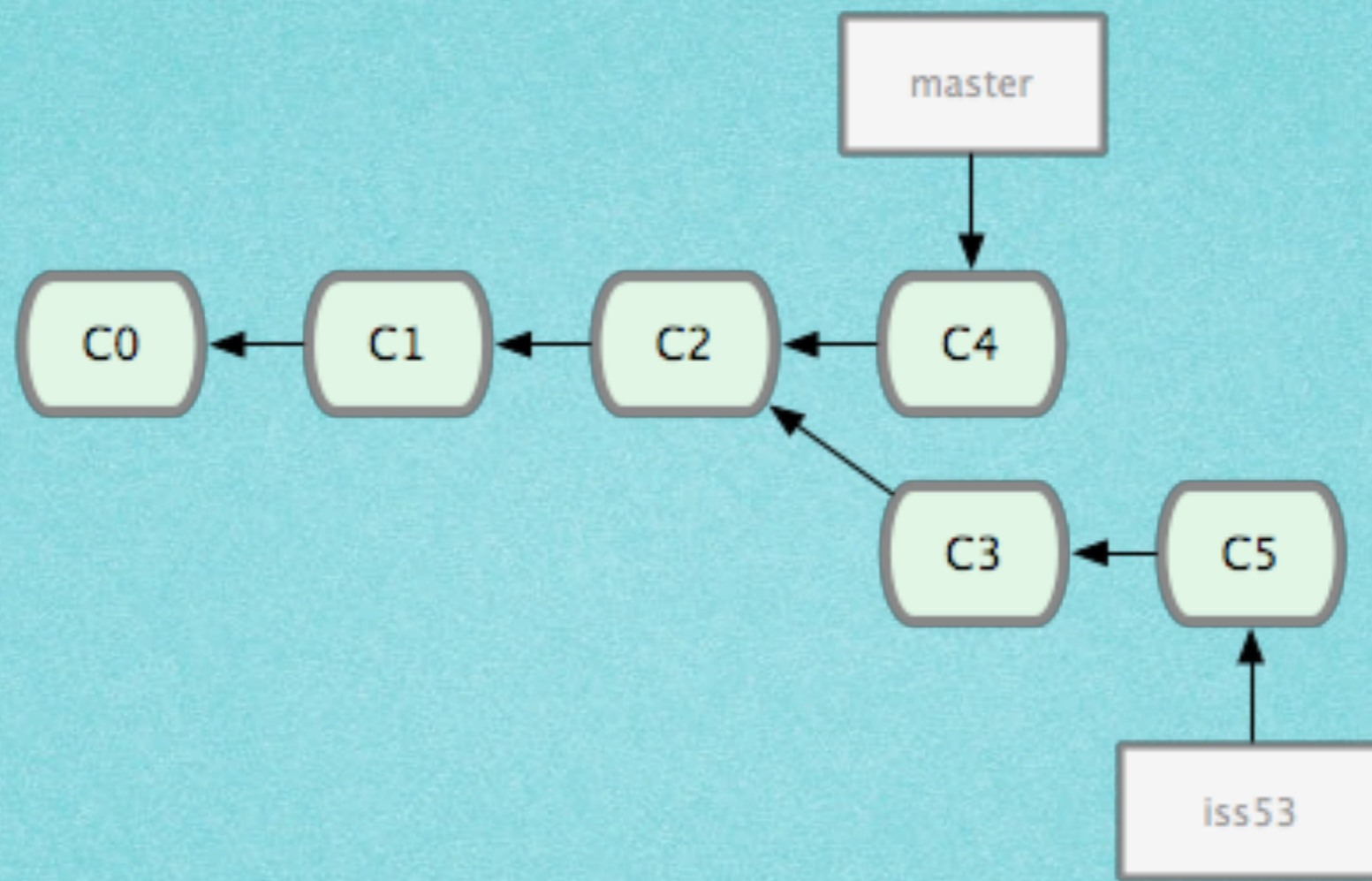




```
git checkout master  
git merge hotfix
```

切到 *master* 之後，把 *master* 和 *hotfix* 「合併」  
因為這種合併只需要 *Commit* 往前推，叫做 ***fast-foward***





# Branch 滿天飛

好，我繼續來寫 *Issue 53* 的 *branch*  
總算寫完了！我要 Merge！



# git 最終的 merge 方式

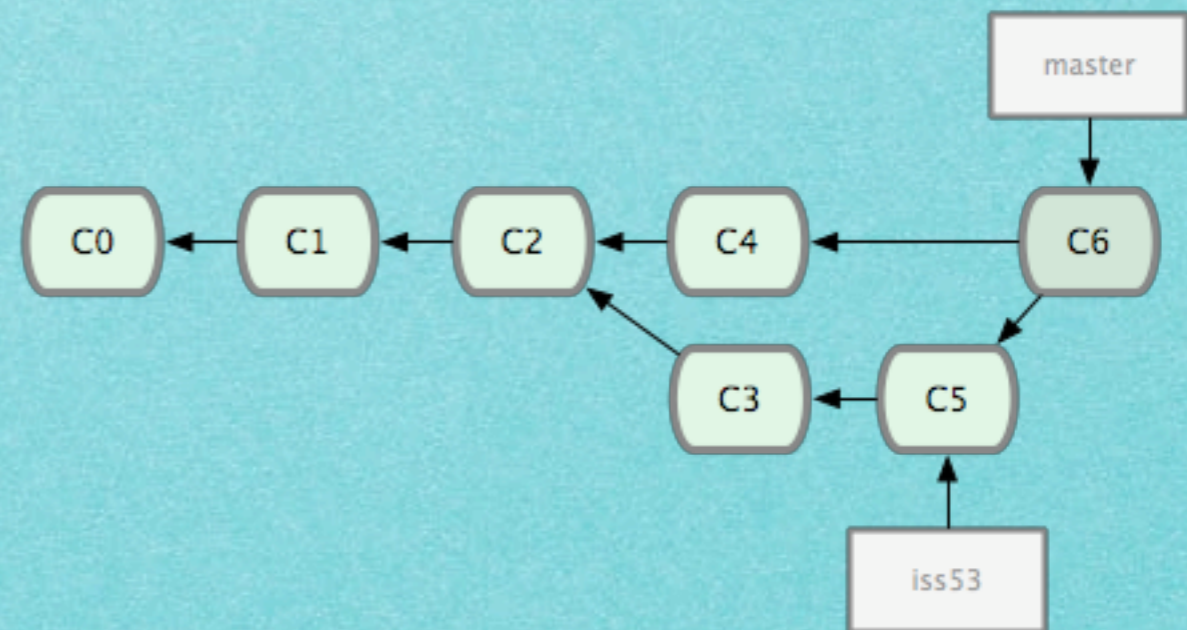
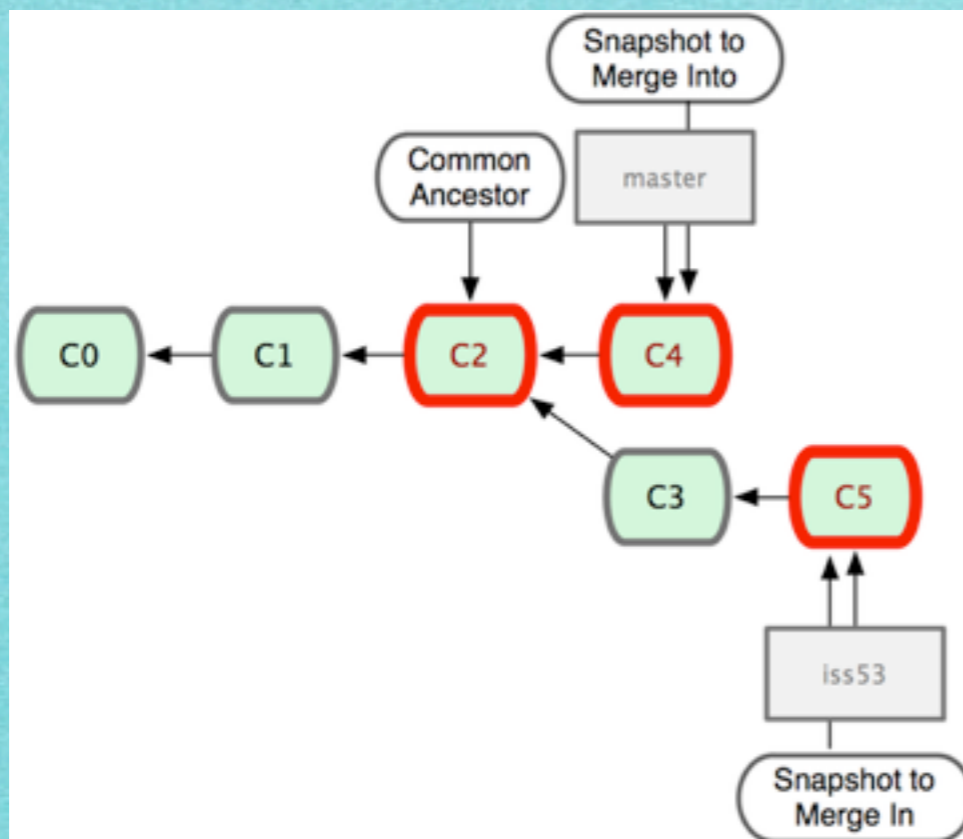
```
$ git checkout master
```

```
$ git merge iss53
```

Merge made by recursive.

```
README | 1 +
```

```
1 files changed, 1 insertions(+), 0 deletions(-)
```





# CONFLICT ! 衝到了 !

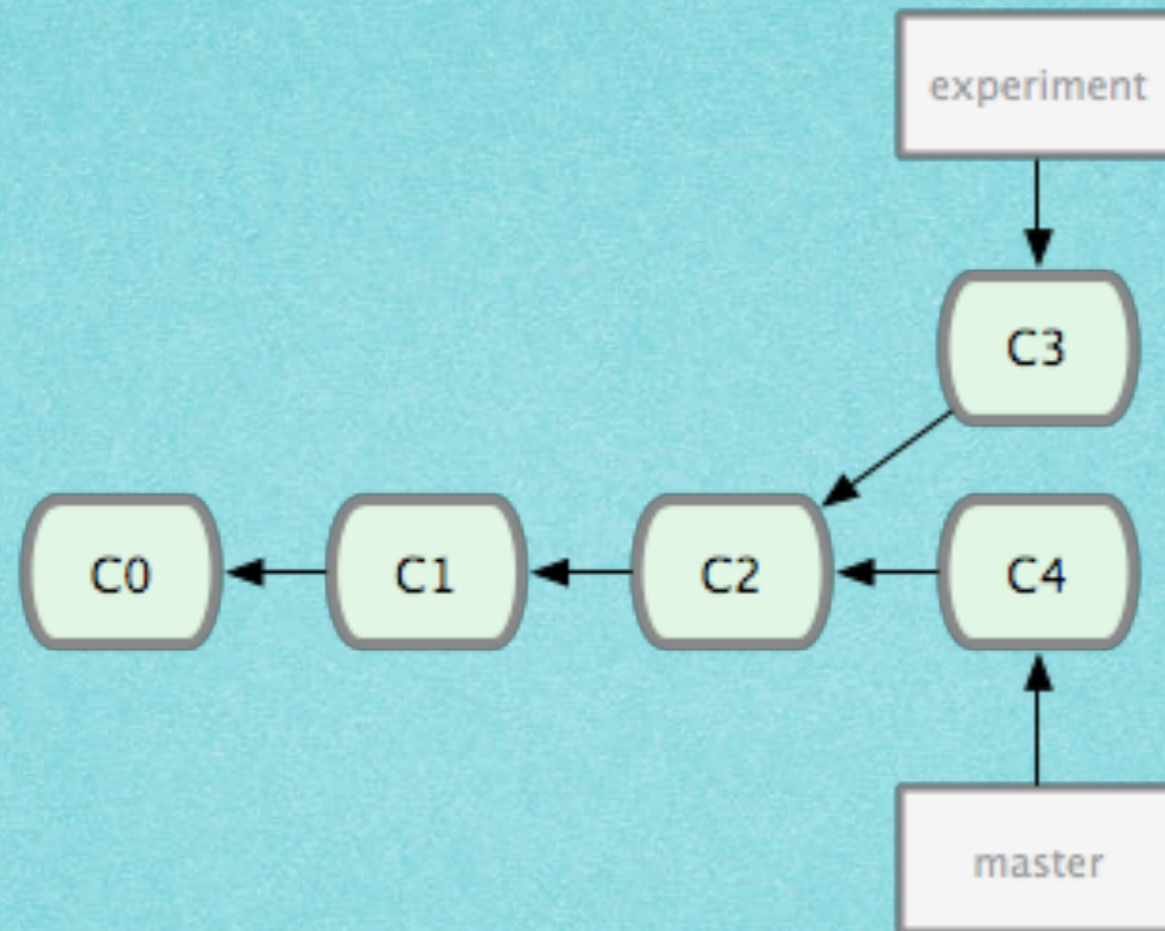
- ▶ 別慌張 !
- ▶ git status 一下看是哪個檔案出問題
- ▶ 到出問題的檔案找問題，手動解決
- ▶ 再檢查 git status 後，用 git commit 手動 Merge (會自動生成 Merge 的 Commit Message)

```
<<<<<<< HEAD:index.html
<div id="footer">contact : email.support@github.com</div>
=====
<div id="footer">
  please contact us at support@github.com
</div>
>>>>>> iss53:index.html
```



*Rebase*

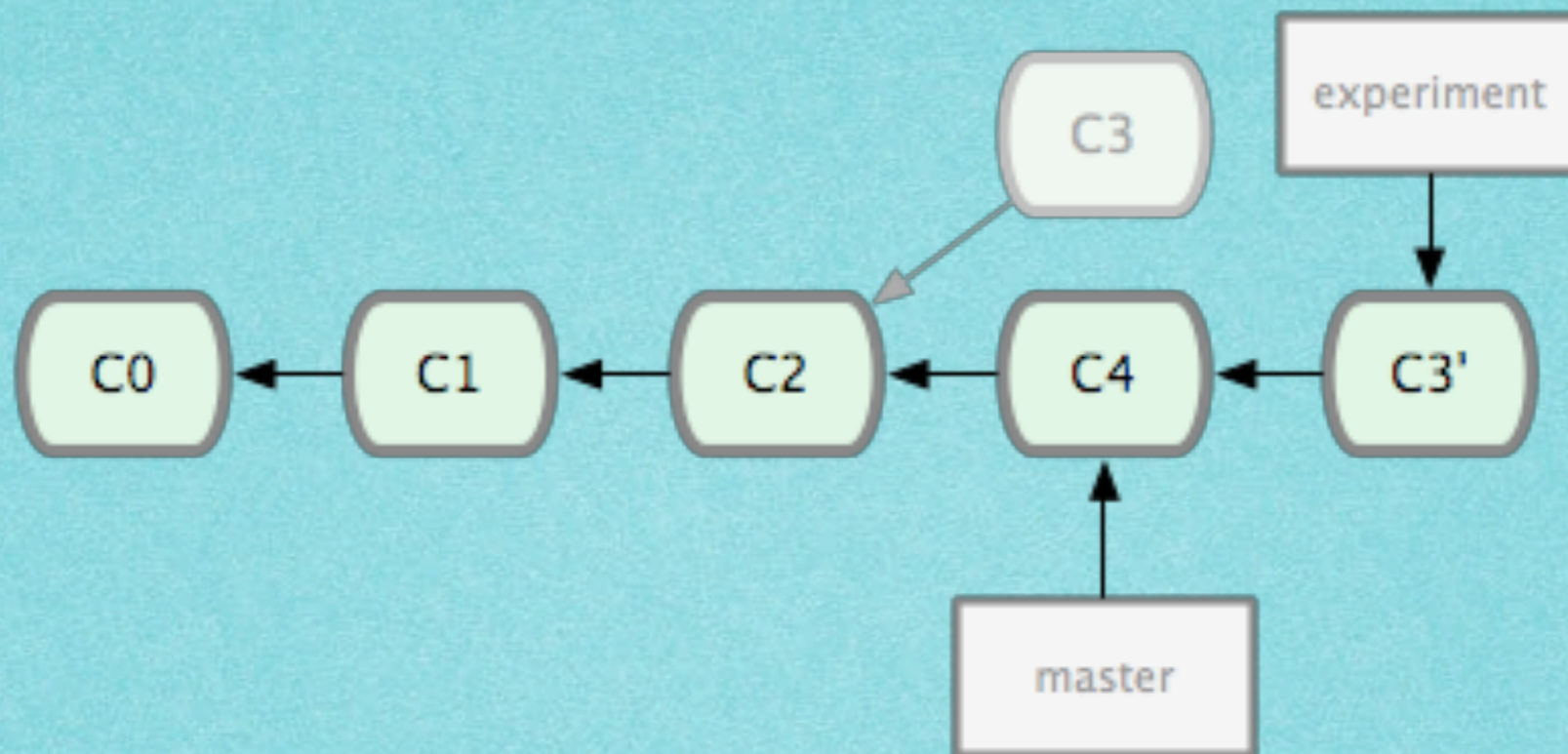




# 回到類似先前的情境

除了 Merge 以外，我有沒有其他的辦法可以處理掉 *experiment branch*?

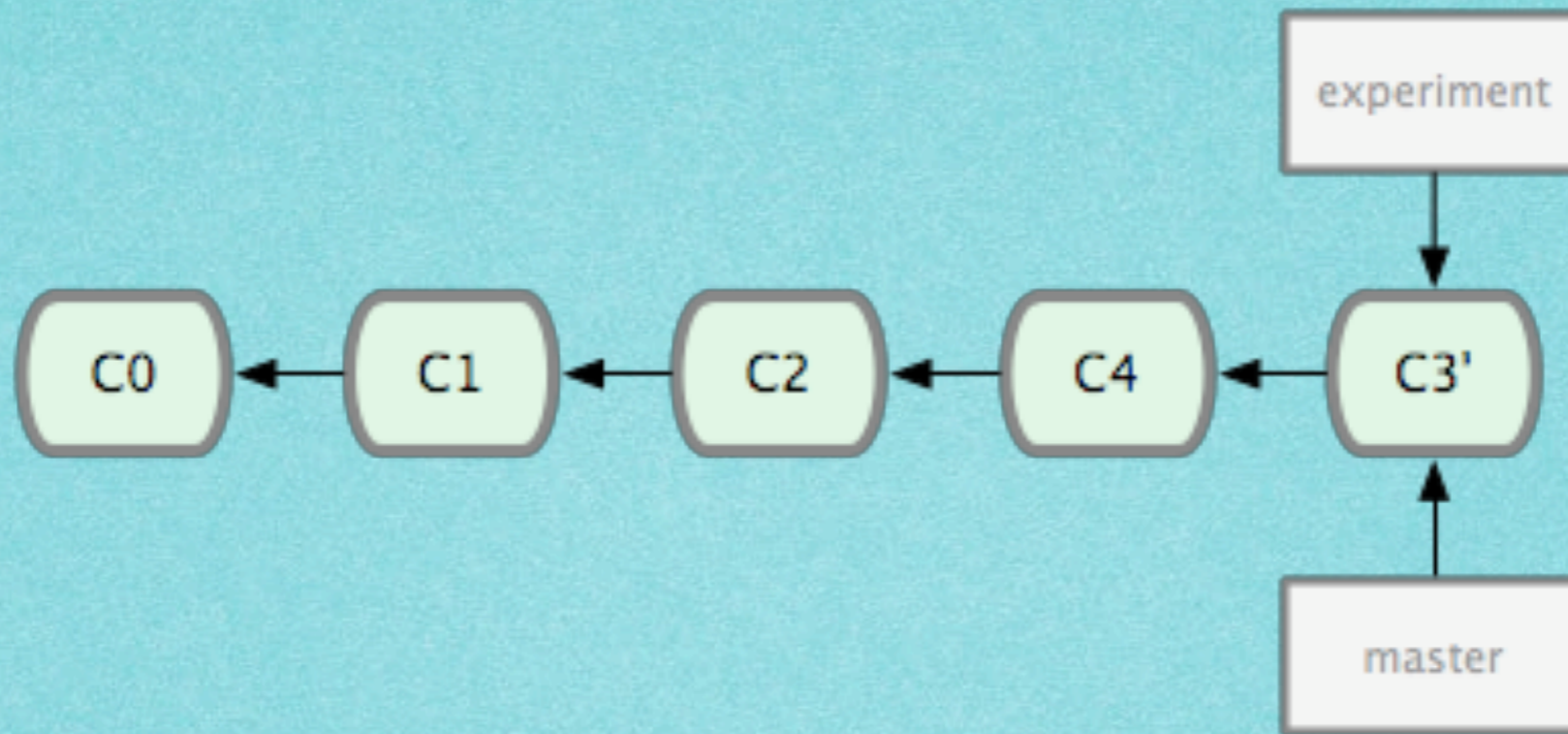




git checkout experiment  
git rebase master

把C3上的變更Rebase到C4之上(**onto C4**)



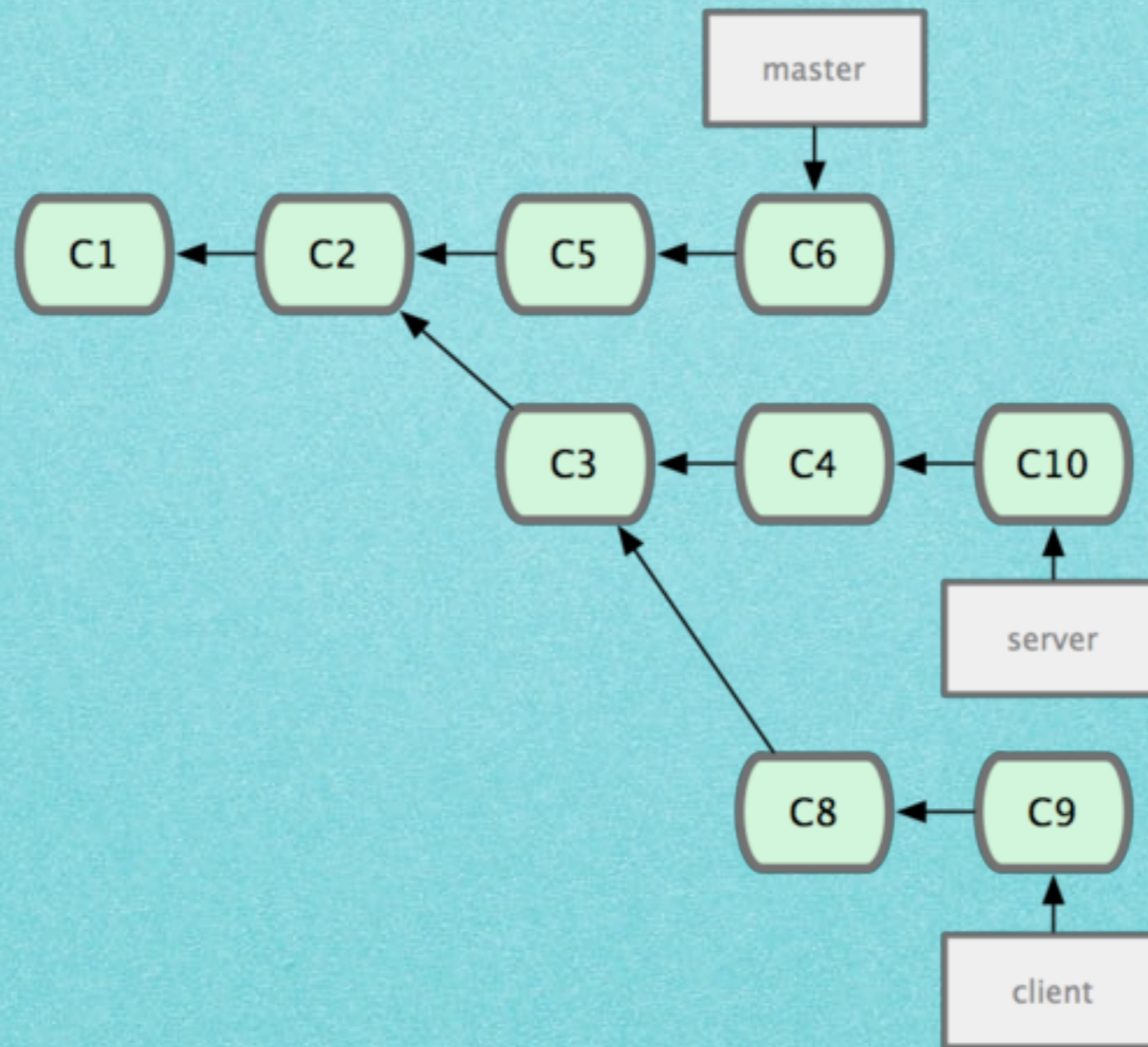


git checkout master  
git merge experiment

*Rebase*過後就可以*fast-forward Merge*了！



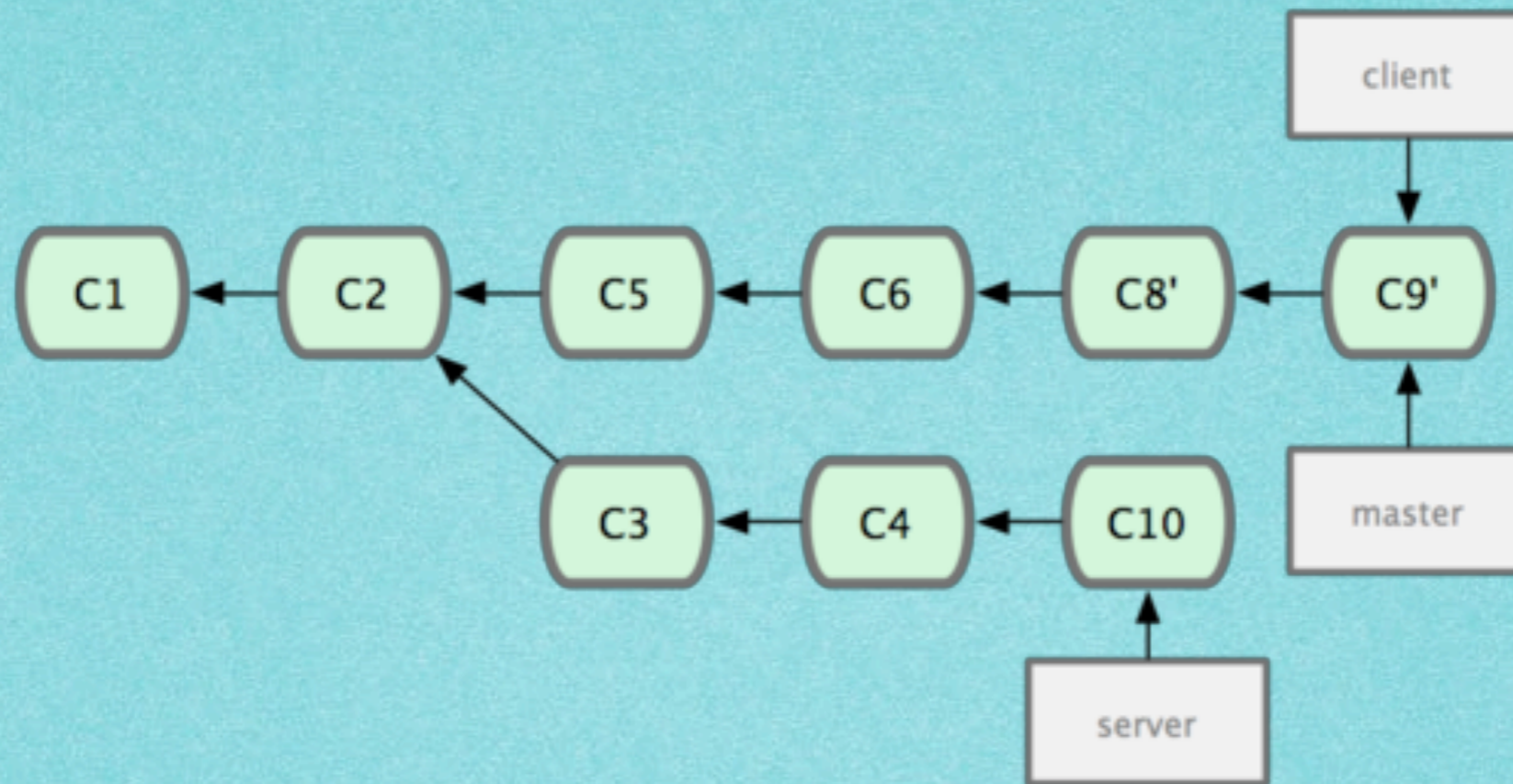
# 當事情更複雜...







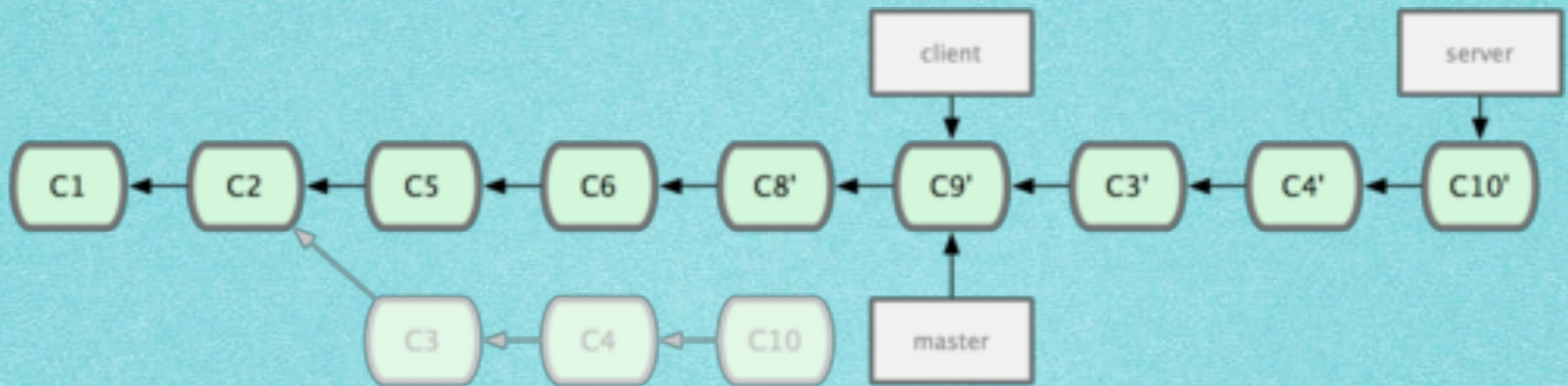




git checkout master  
git merge client

對client作fast-forward Merge

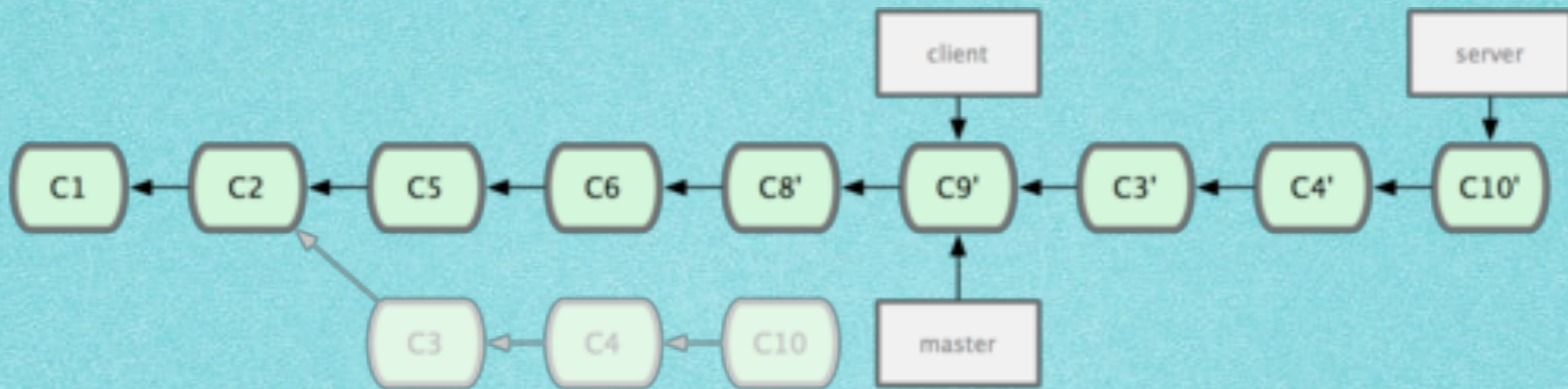




git rebase master server

直接指定要將哪個 *Branch* 的更動 (*server*)  
重新在哪個 *Branch* 作過一遍 (*master*)

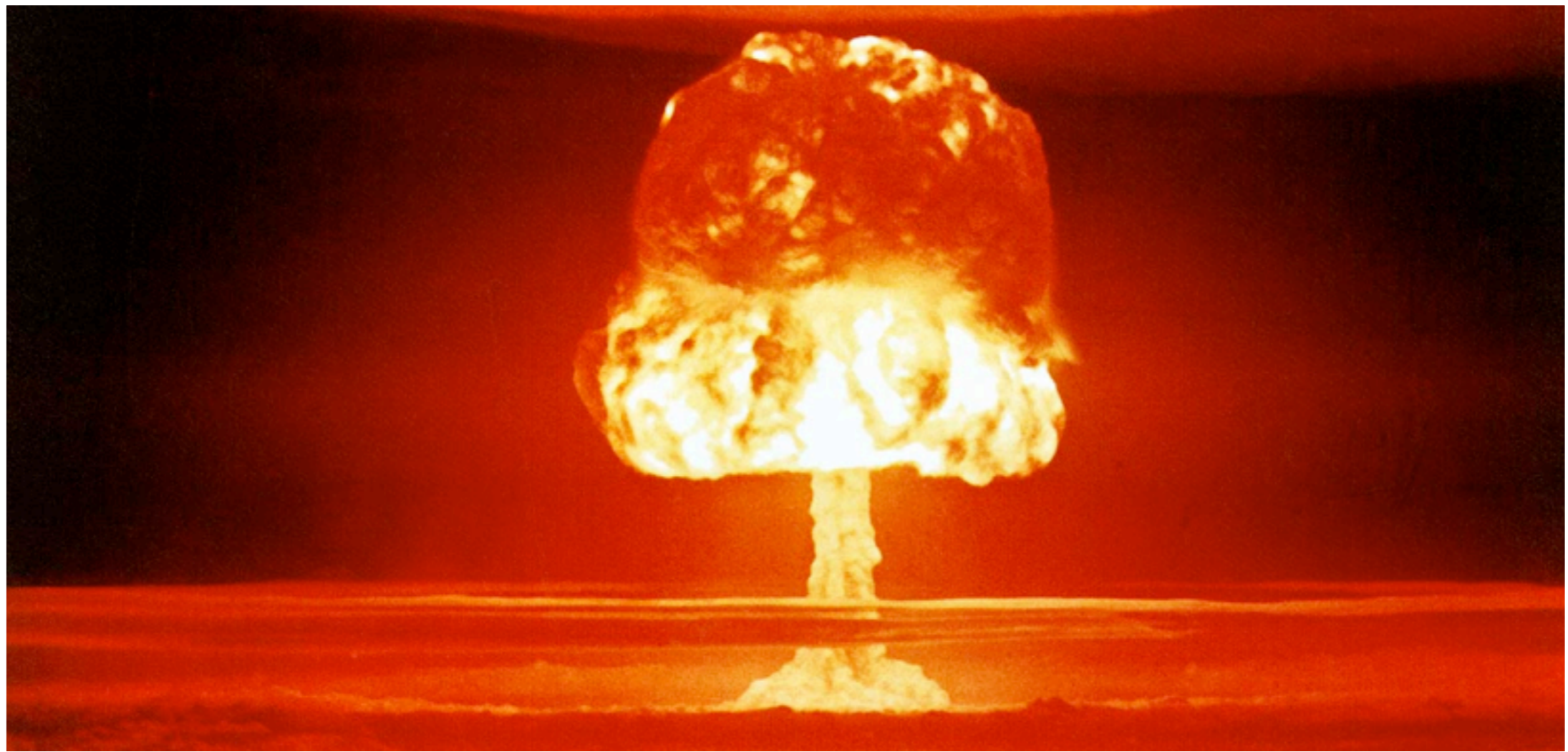




```
git checkout master  
git merge server  
git branch -d client  
git branch -d server
```

通通都Merge完之後  
把兩條不要的Branch砍掉囉 :)





「千萬不要」對已經Push的東西  
作Rebase！

除非你想成為害群之馬。

Pic: [http://commons.wikimedia.org/wiki/File:Castle\\_Romeo.jpg](http://commons.wikimedia.org/wiki/File:Castle_Romeo.jpg), Public Domain, United States Department of Energy

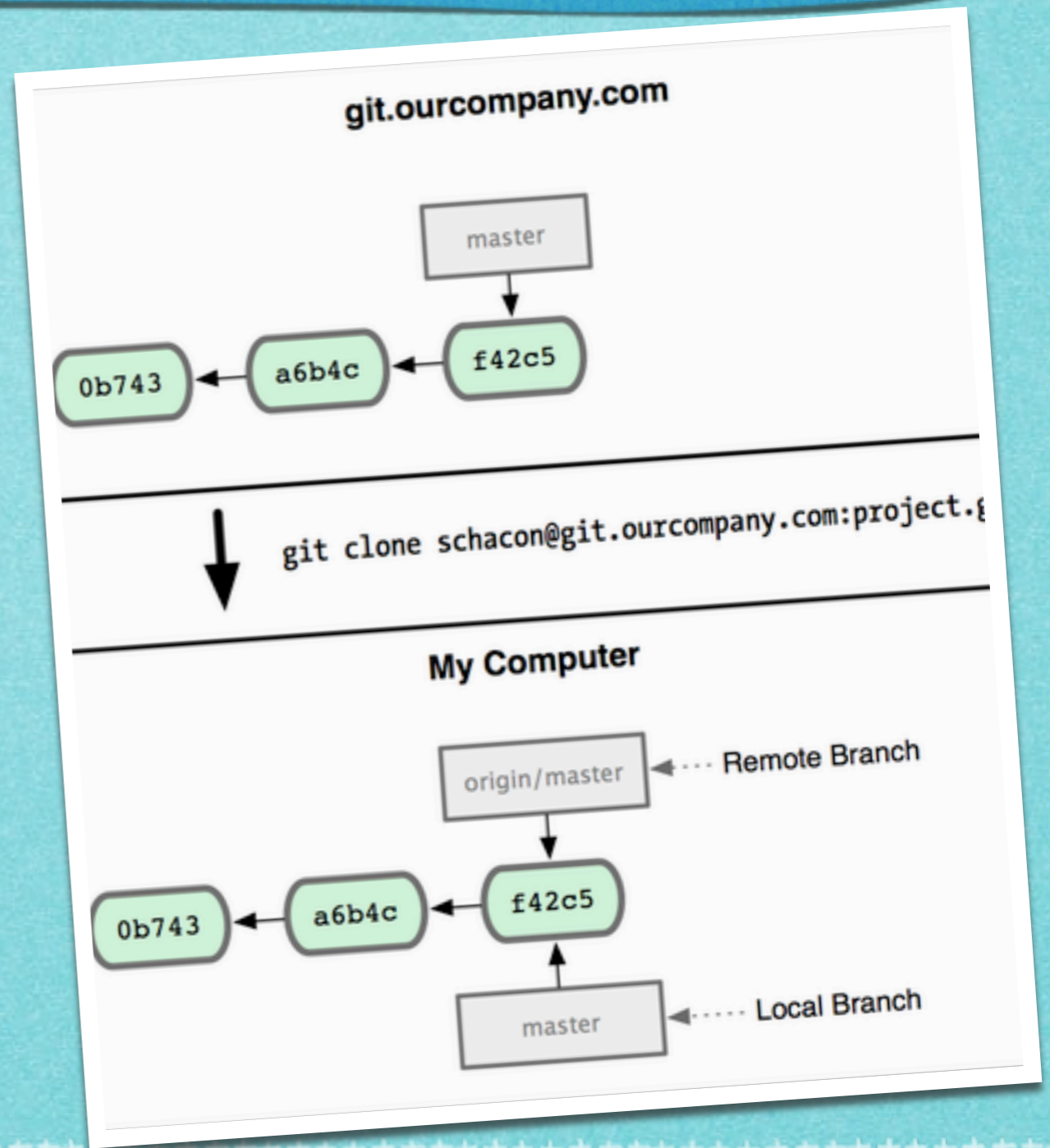


# *Remote Branches*



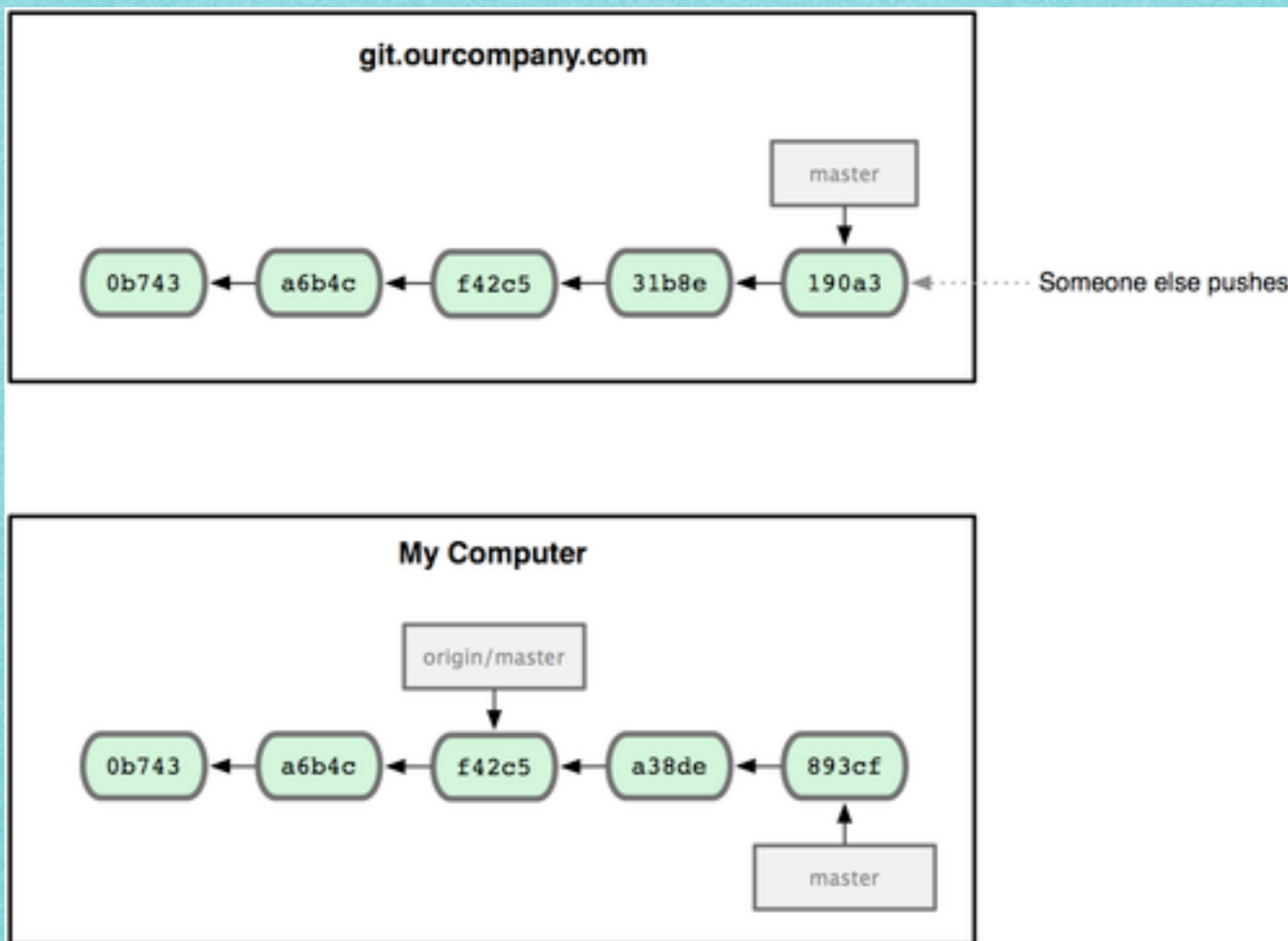
# Remote Branch

- ▶ 記錄「遠端套件庫」上的Branch狀態
- ▶ origin上的master分支就是origin/master
- ▶ 所以git clone會跑出origin/master和master





# 當你與Remote Branch 同時改了東西...



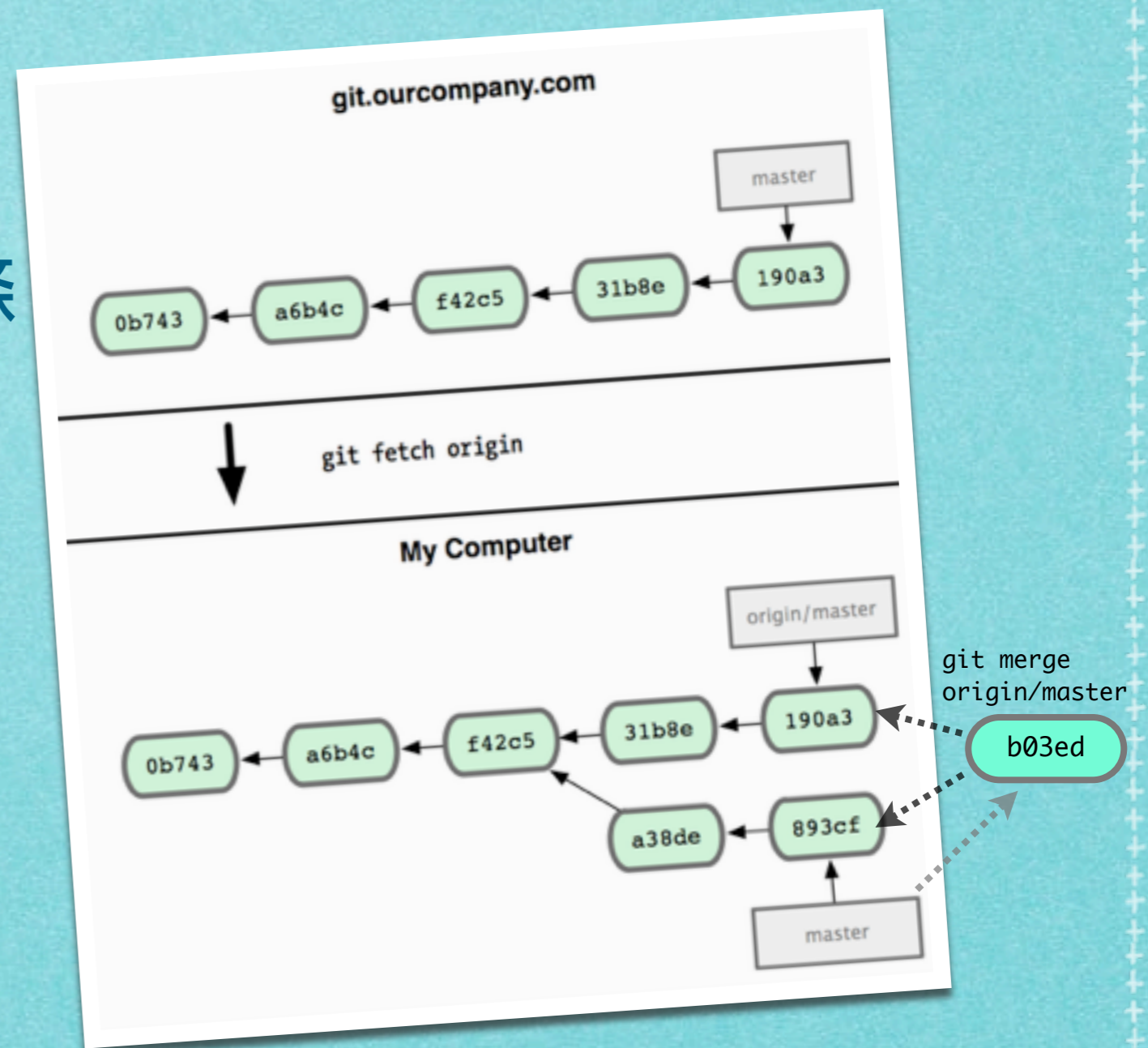


# Pull = Fetch + Merge

▶ 把Origin作Fetch之後，兩邊同時有更動，因此會變成兩條分支

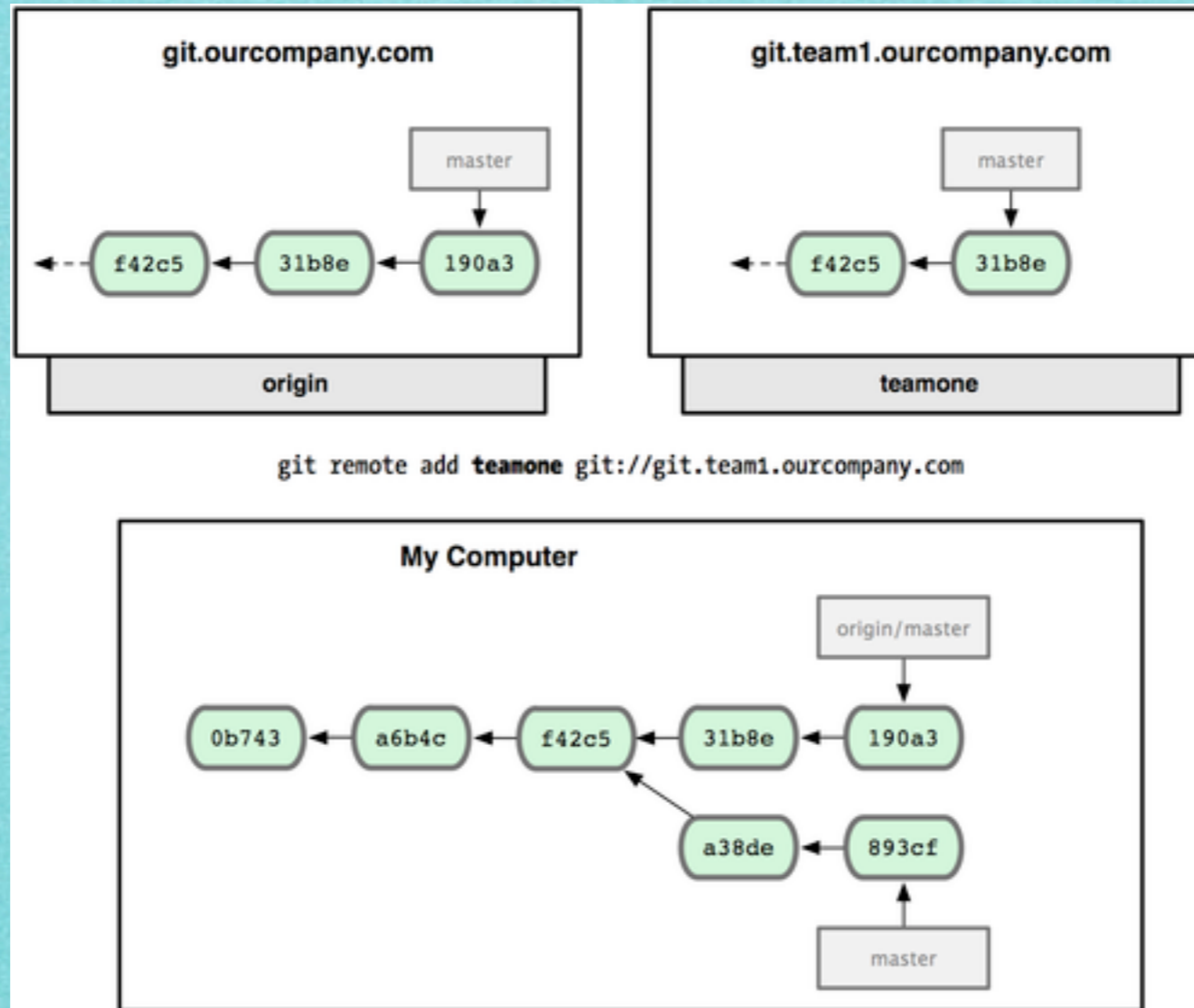
▶ 通常的作法是，用Merge合併回一條

▶ 所以會有git pull：  
git pull origin =  
git fetch origin 之後  
git merge origin/master





# 新增一個Remote





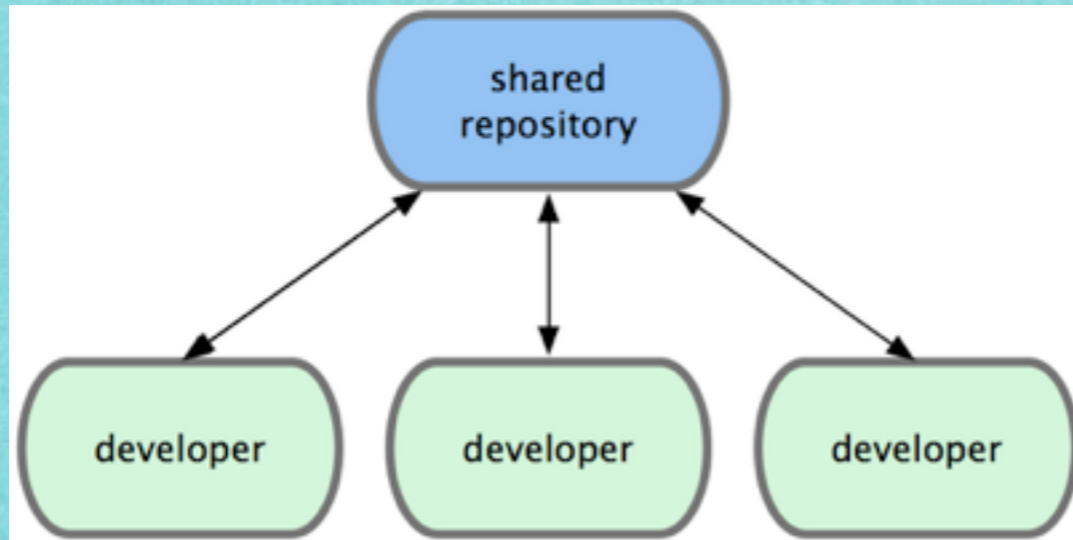




# *Distributed Workflows*

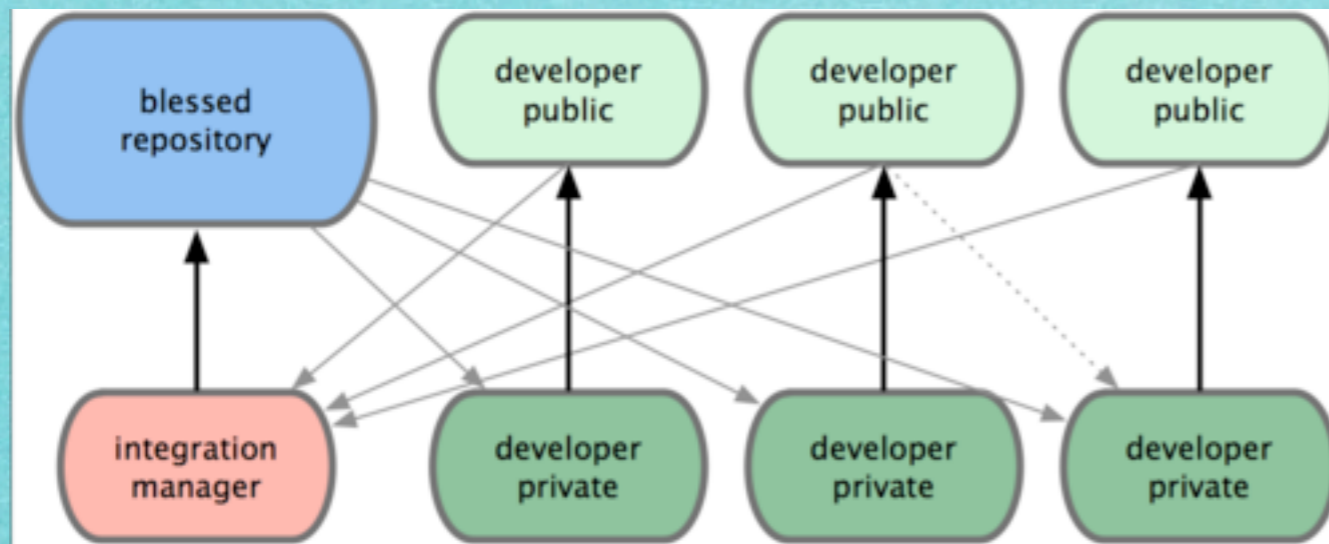


# 開發方式任你選擇

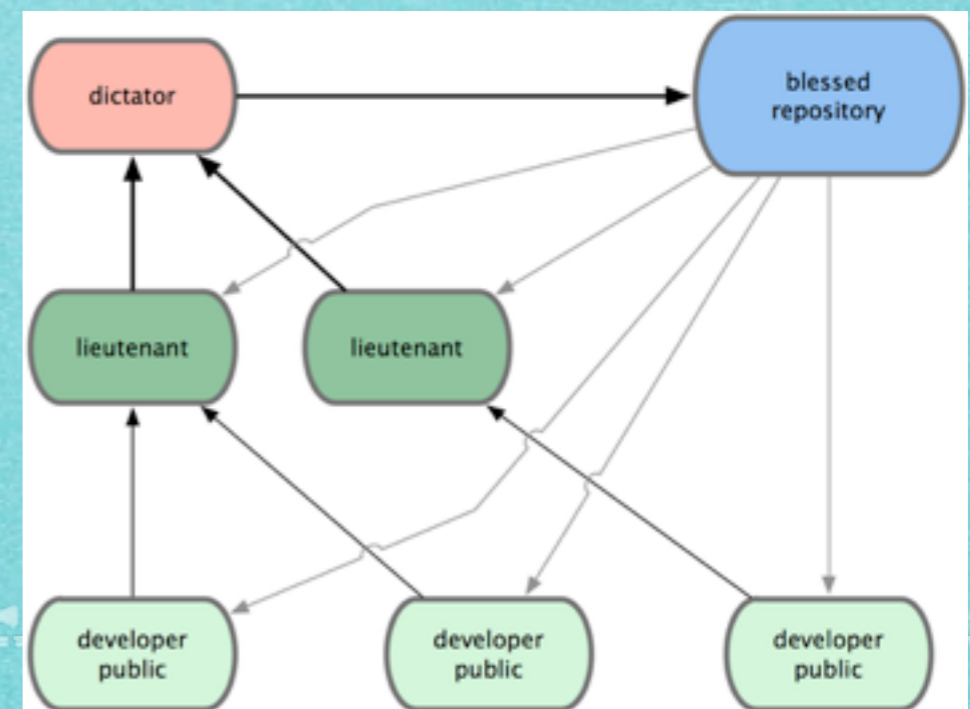


中央式的開發

整合管理員



司令官和副手



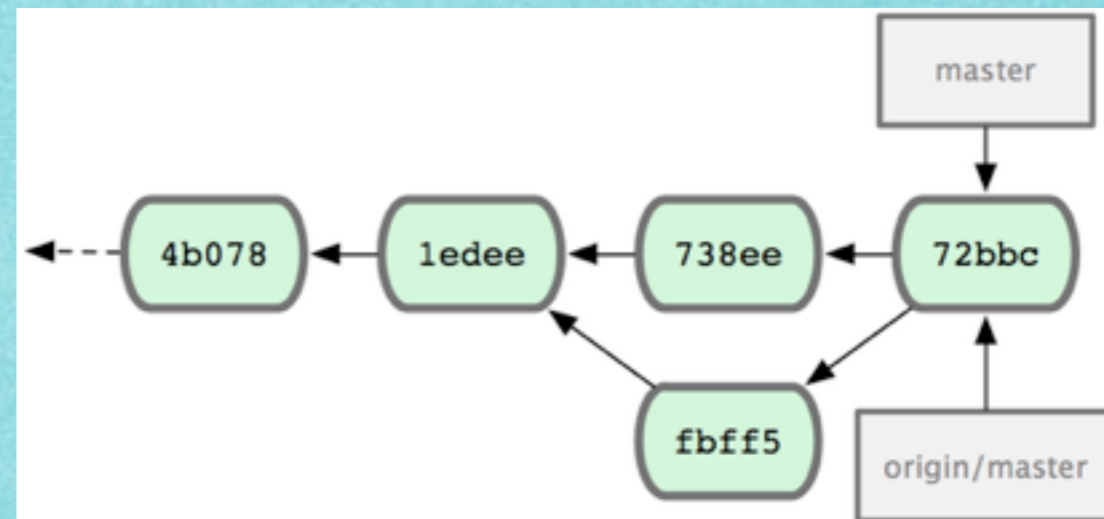
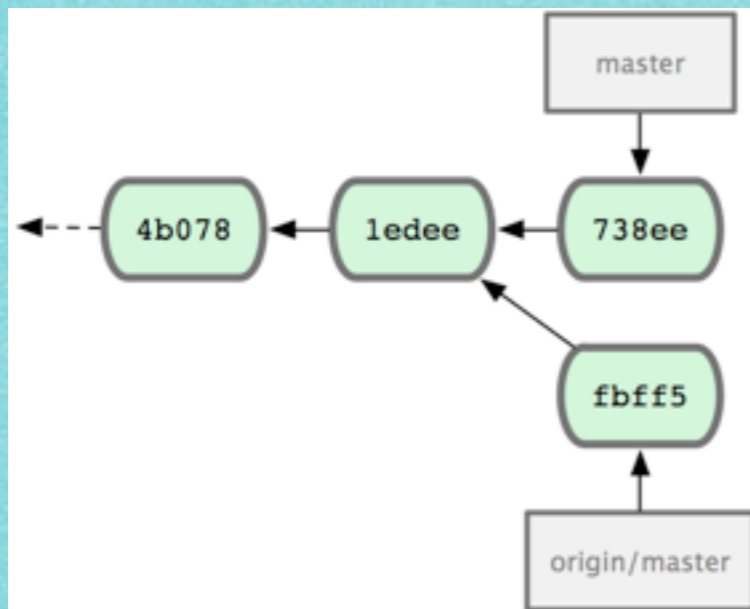


# 一些基本守則

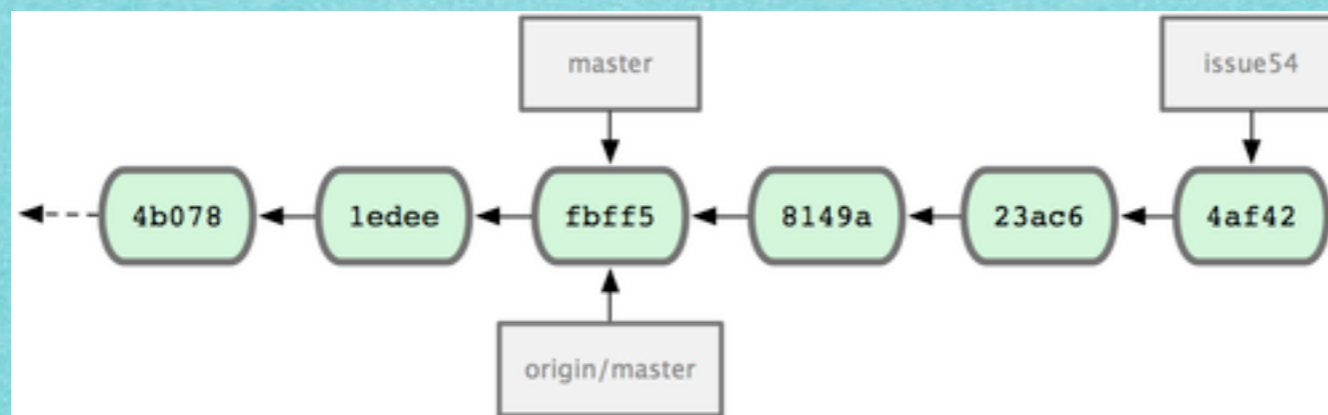
- ▶ 當你要Commit/送交Patch時：
  - ▶ 用`git diff --check`檢查行尾有沒有多餘的空白
  - ▶ 「每個Commit只改一件事情」  
如果一個檔案有多個變更，用`git add --patch`  
只選擇檔案中的部分變更進Stage
  - ▶ 寫清楚Commit Message！



# 私人小團體協作



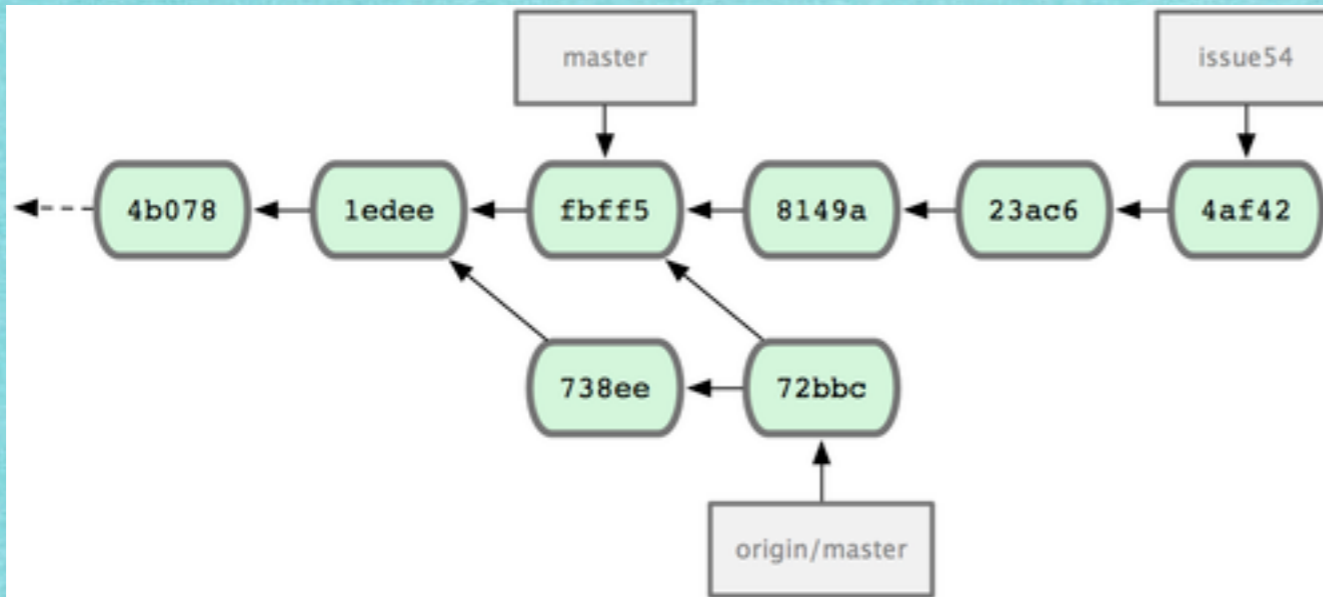
**John** 改了738ee之後Fetch跑出兩條，於是Merge起來，一切正常後Push



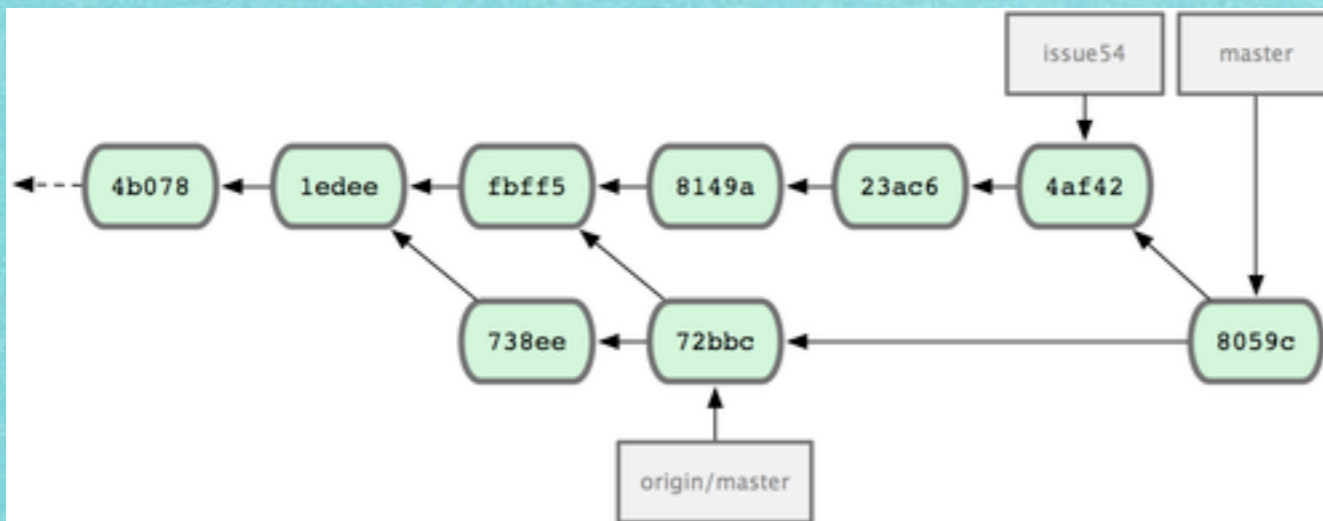
**Jessica** 開了一個Branch專心來處理 Issue 54 的臭蟲



# Jessica會怎麼做？



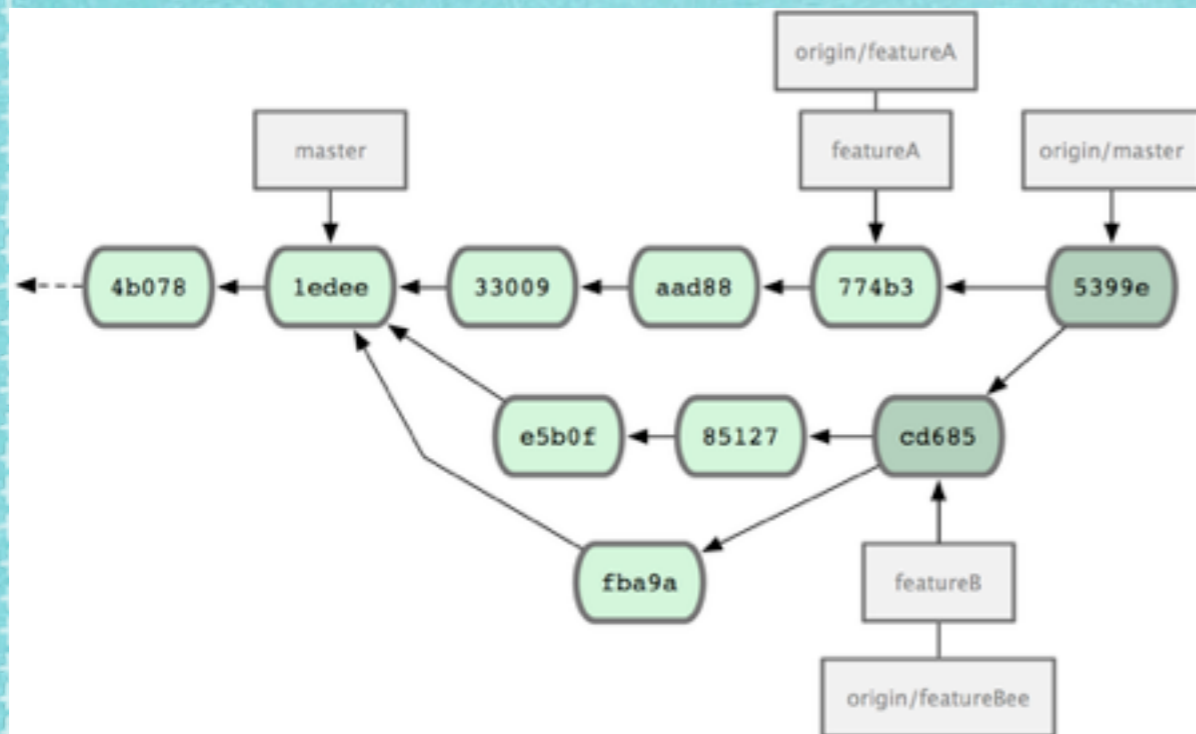
Fetch之後，Jessica發現John有更動，檢查過後，先合併issue54 (Fast-forward)：  
git checkout master  
git merge issue54



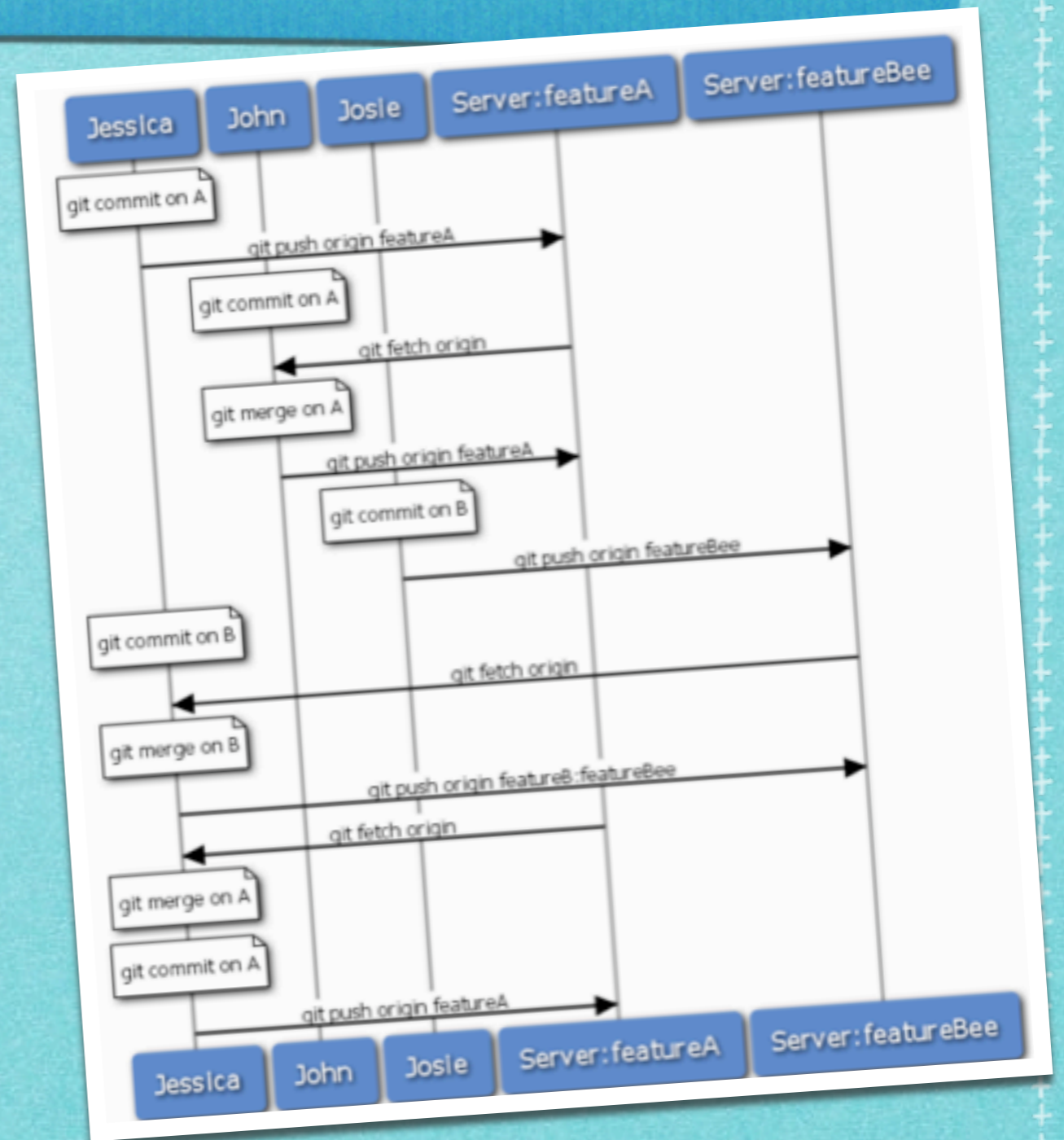
然後合併origin/master：  
git merge origin/master  
就可以快樂Push了！



# 規模較大私人團體的協作

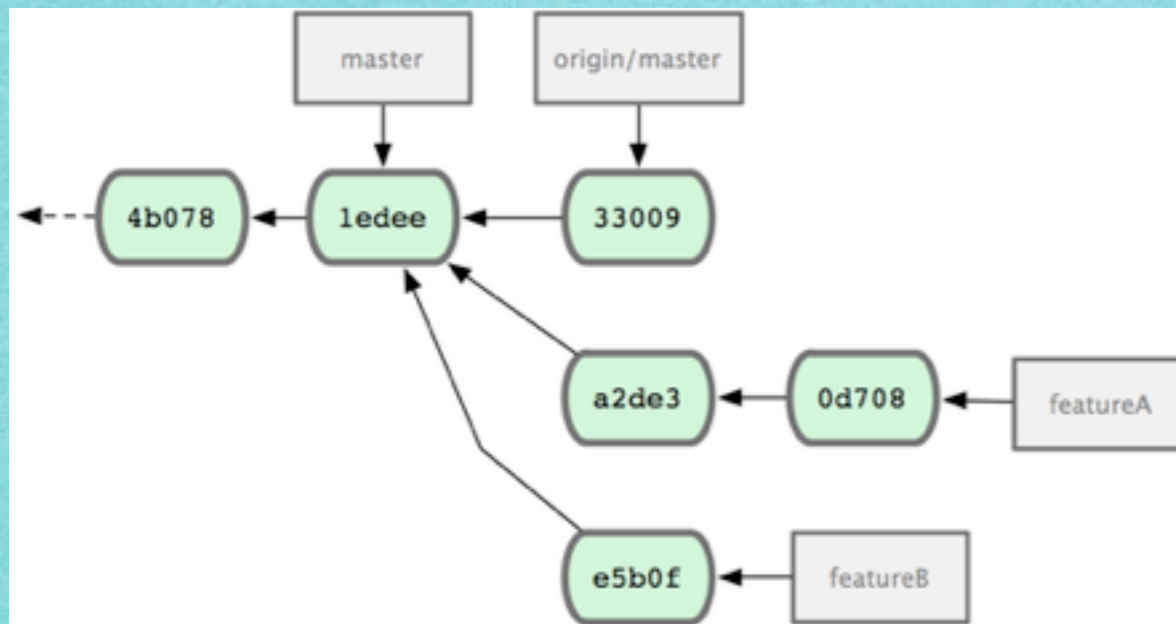


三個人一起工作  
開兩個Branch寫兩種功能  
最後把一切Merge起來



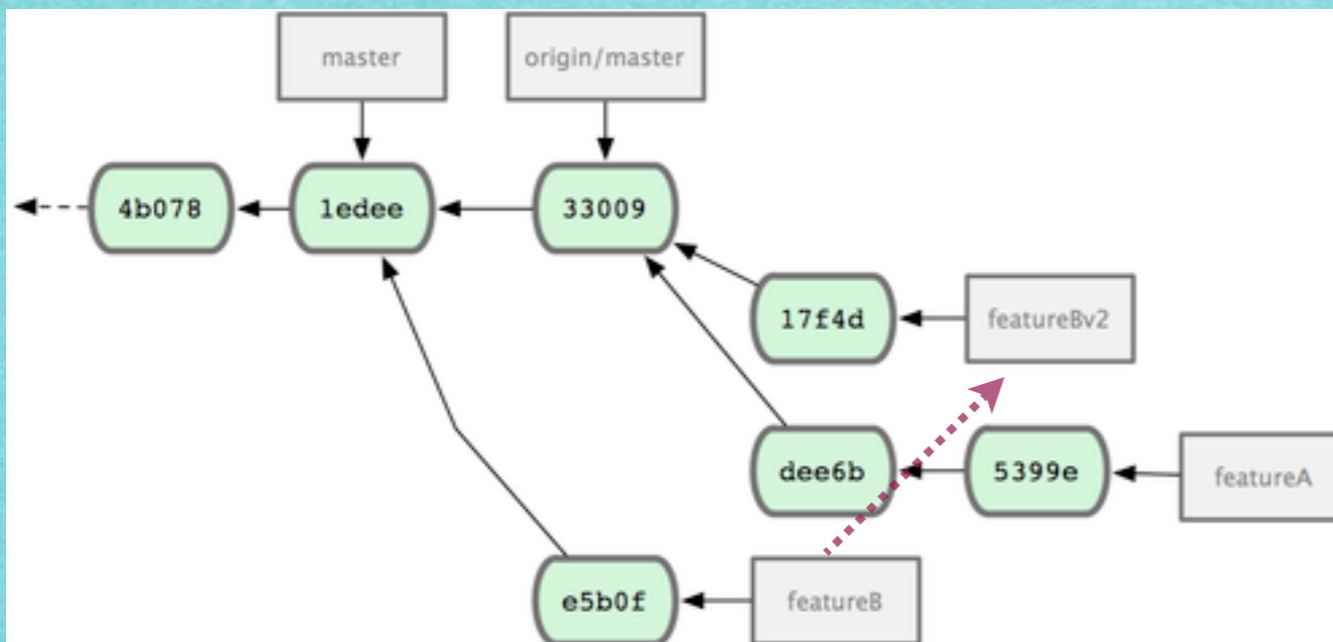


# 公開的小型協作



開了A、B兩個Branch之後  
先用Rebase把A組好：

```
$ git checkout featureA  
$ git rebase origin/master  
$ git push -f myfork featureA
```



再用「奇妙Merge」  
把B組回去：

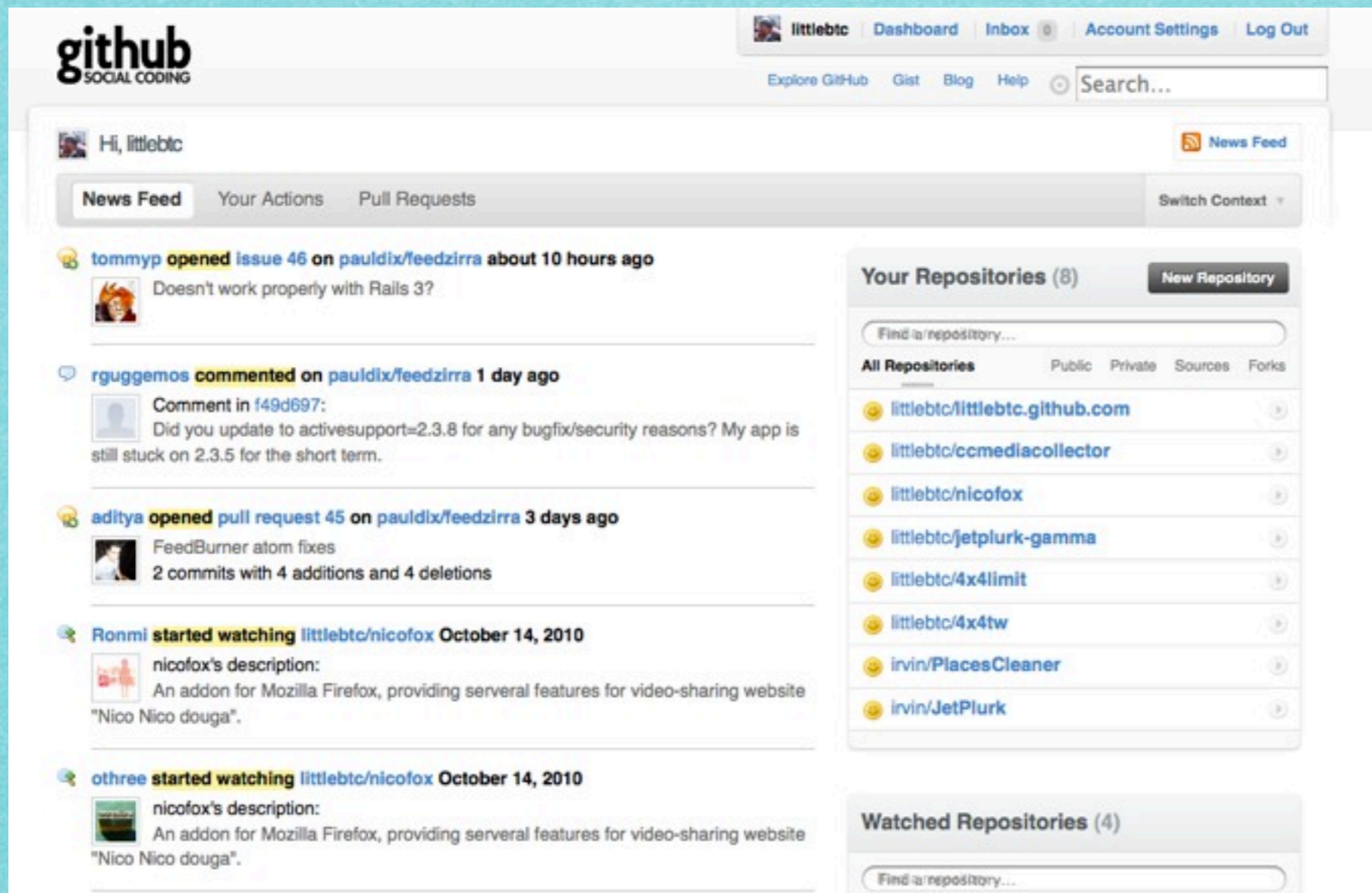
```
$ git checkout -b featureBv2 origin/master  
$ git merge --no-commit --squash featureB  
$ (change implementation)  
$ git commit  
$ git push myfork featureBv2
```



*... Still a lot to learn!*



# github 很好玩！



The screenshot shows the GitHub user interface for a user named 'littlebtc'. At the top, there's a navigation bar with 'Dashboard', 'Inbox', 'Account Settings', and 'Log Out'. Below that, a search bar and a 'News Feed' button are visible. The main content area is divided into two columns. The left column shows a 'News Feed' with several items: 'tommy opened issue 46 on pauldix/feedzorra about 10 hours ago', 'rguggemos commented on pauldix/feedzorra 1 day ago', 'aditya opened pull request 45 on pauldix/feedzorra 3 days ago', and two items where 'Ronmi' and 'othree' started watching 'littlebtc/nicofox' on October 14, 2010. The right column shows 'Your Repositories (8)' with a list of repositories including 'littlebtc/littlebtc.github.com', 'littlebtc/ccmediacollector', 'littlebtc/nicofox', 'littlebtc/jetplurk-gamma', 'littlebtc/4x4limit', 'littlebtc/4x4tw', 'irvin/PlacesCleaner', and 'irvin/JetPlurk'. Below that, there's a section for 'Watched Repositories (4)'.



...而且對公開的 Open Source 專案免費！



# 了解更多

- ▶ *Pro Git* 線上電子書：<http://progit.org/>  
這份投影片大部分的內容都出於此處，是很好的 Git 線上說明！
- ▶ ihower 的 Git 文章系列：<http://ihower.tw/blog/archives/category/git>