給我30分鐘
教你學會一套git的好菜

腹黒い茶 (*Haraguroicha Hsu*)

本簡報是基於Littlebtc的「寫給大家的 Git 教學」與於Scott Chacon的「Pro Git」電子書所修改的衍生版本

What is Version Control?
What is Git?
Why we needs that?

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
  - 刪了東西，不會救不回來
- 「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

這裡先不介紹其他版本控制

從 *Git* 的誕生講起

Linux Kernel 是一切的開端

(http://en.wikipedia.org/wiki/Linux_kernel, Wikipedia contributors, CC-BY-SA)

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 CC-BYSA-3.0 & GFDL)

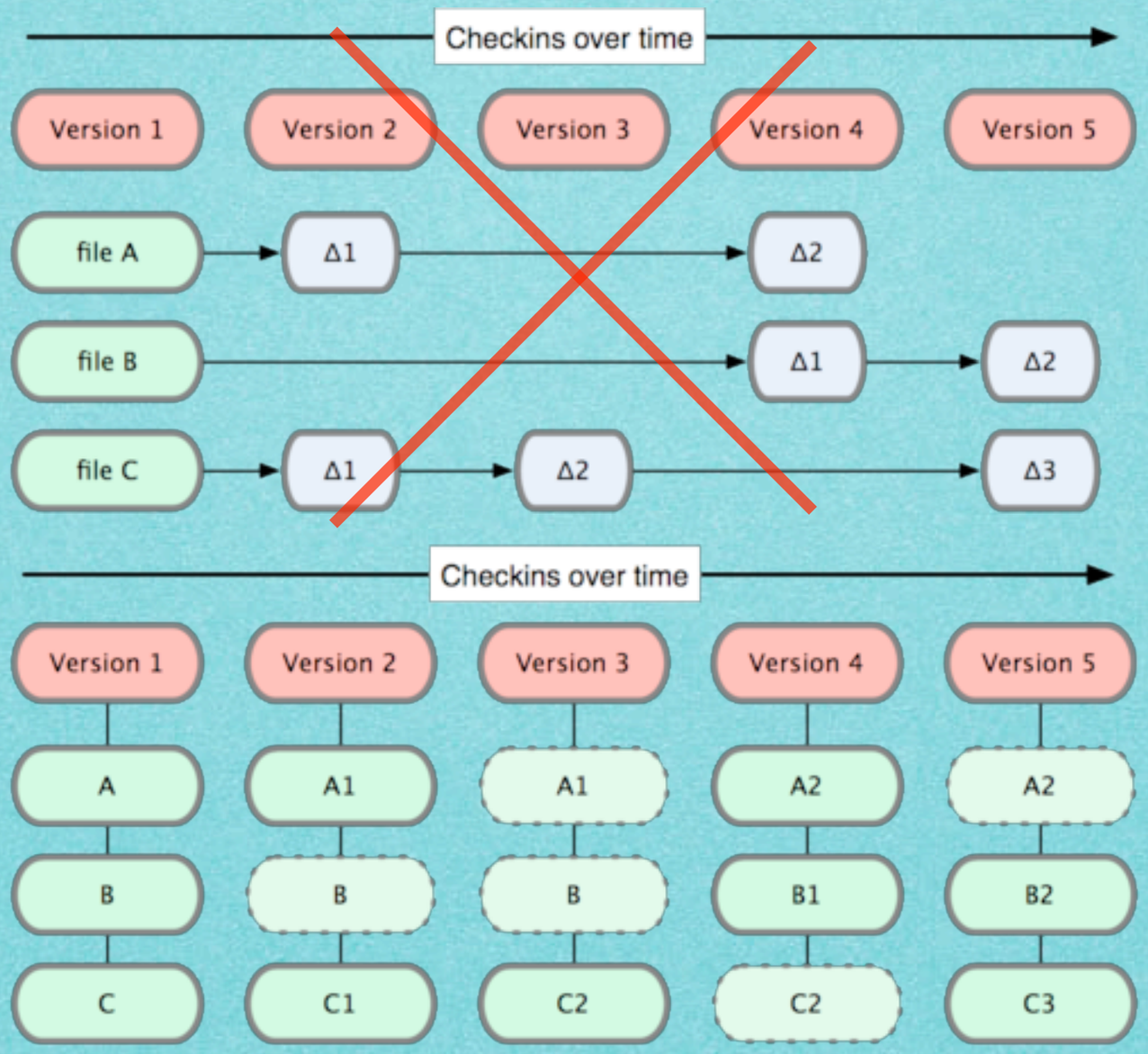
6

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

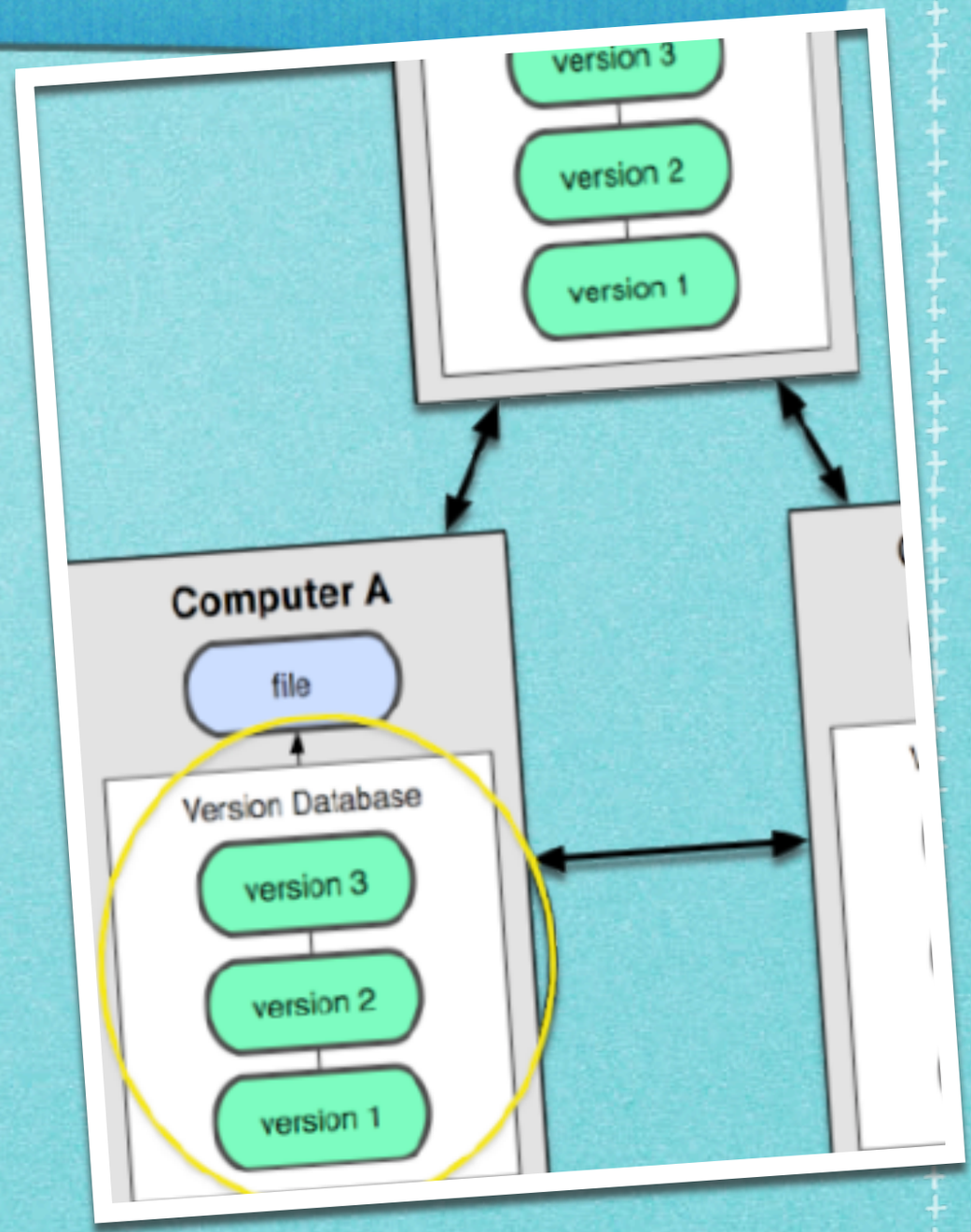
Git不是記錄「檔案的差異」
而是記錄「檔案的快照」



9

Git 幾乎什麼事都在本機進行！

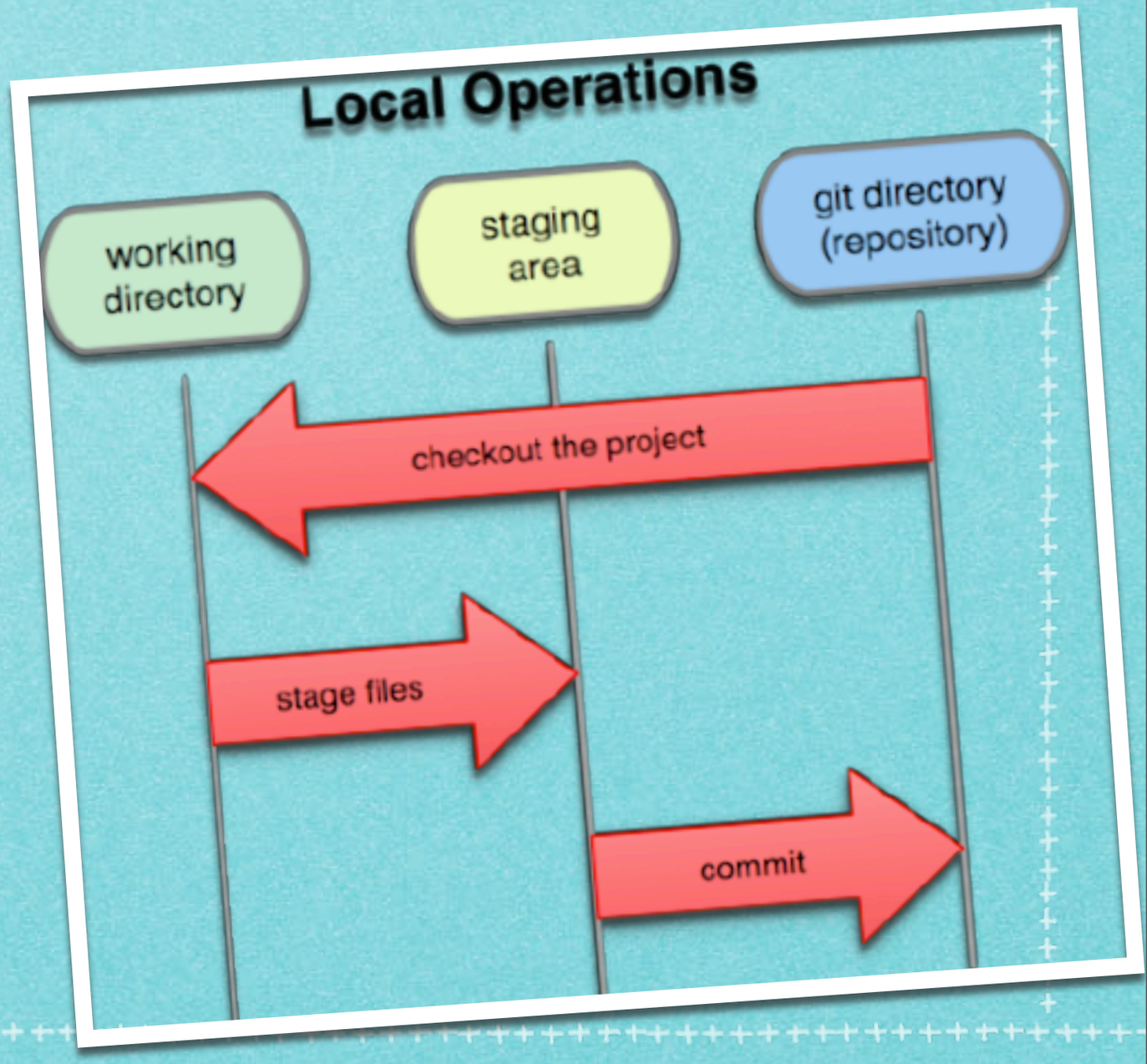
- 你不只有一份完整的資料庫.....
- 閱讀版本歷史、提交變更這些動作都可以在本機進行
- 不需要網路連線也能單獨工作！



10

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



11

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」

- 找一個空的或建立一個新的資料夾，然後切換到該資料夾內執行命令：

```
$ git init
```

- 套件庫所需要的資訊都會放在.git的隱藏資料夾中

13

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'
```


14

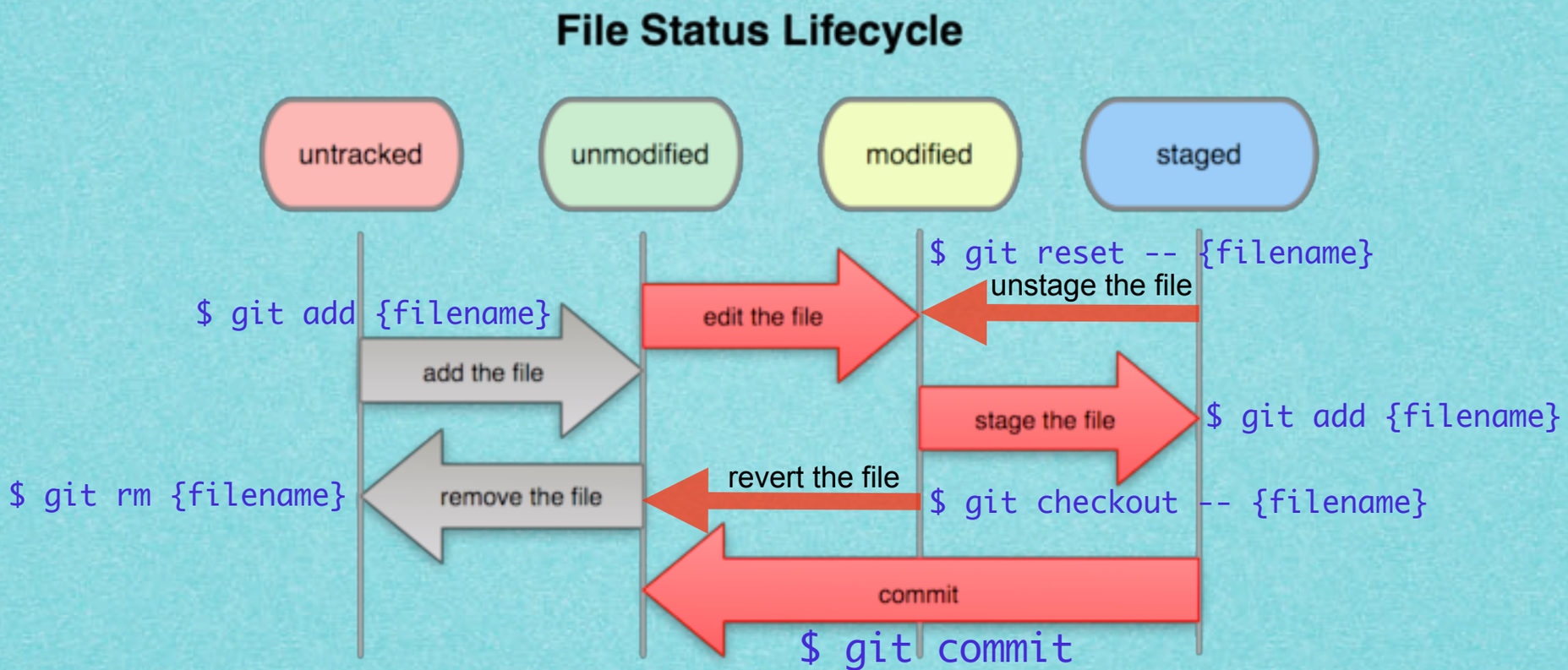
複製別人的 Repository

- Clone的專案會放在目前路徑下的新目錄。

```
$ git clone http://github.com/jquery/jquery.git
Cloning into 'jquery'...
remote: Counting objects: 27360, done.
remote: Compressing objects: 100% (7244/7244), done.
remote: Total 27360 (delta 20112), reused 26606 (delta 19519)
Receiving objects: 100% (27360/27360), 14.18 MiB | 177 KiB/s, done.
Resolving deltas: 100% (20112/20112), done.
```


15

File Status



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

16

git status

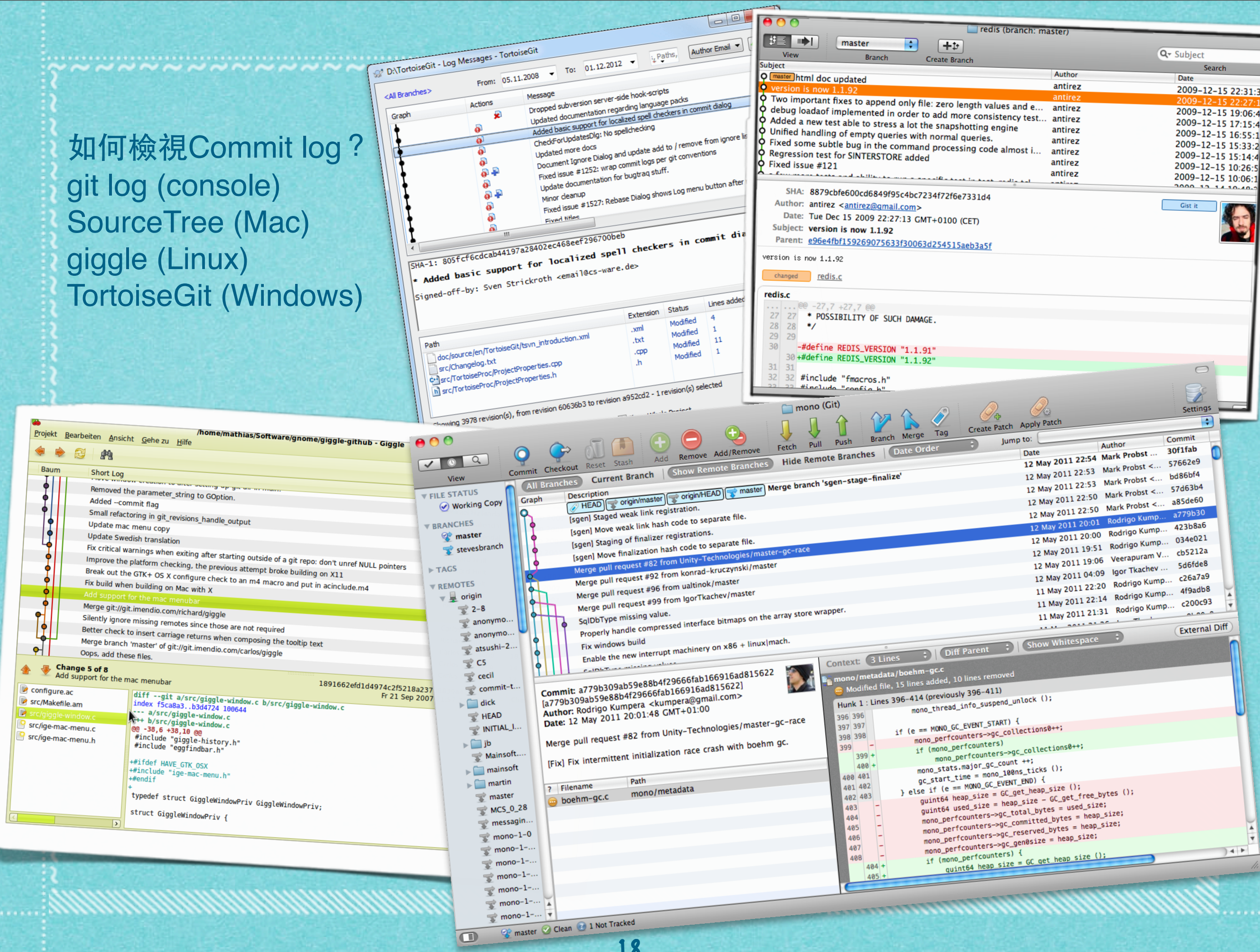
git — bash — 80×30 — ⌘1

```
[20:35:13]chaochan@Haraguroicha-MacBook-Air:git M(master)$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add/rm <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       deleted:    gitCook.key/QuickLook/Preview.pdf
#       modified:   gitCook.key/index.apxl.gz
#       deleted:    gitCook.key/theme-files/Scrapbook_photo_low-1.png
#       modified:   gitCook.key/thumbs/mt0-0.jpg
#       deleted:    gitCook.key/thumbs/mt0-11.jpg
#       modified:   gitCook.key/thumbs/mt0-12.jpg
#       modified:   gitCook.key/thumbs/mt0-15.jpg
#       modified:   gitCook.key/thumbs/mt0-17.jpg
#       deleted:    gitCook.key/thumbs/mt0-19.jpg
#       modified:   gitCook.key/thumbs/mt0-20.jpg
#       deleted:    gitCook.key/thumbs/mt0-21.jpg
#       deleted:    gitCook.key/thumbs/mt0-22.jpg
#       modified:   gitCook.key/thumbs/mt0-6.jpg
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       391px-Linus_Torvalds.jpeg
#       gitCook.key/391px-Linus_Torvalds.jpeg
#       gitCook.key/droppedImage-1.tiff
#       gitCook.key/droppedImage-3.tiff
#       gitCook.key/droppedImage-4.tiff
#       gitCook.key/droppedImage-5.tiff
#       gitCook.key/droppedImage-6.tiff
```


17

如何檢視Commit log？

git log (console)
SourceTree (Mac)
giggle (Linux)
TortoiseGit (Windows)



改爛了...!?

- 改變最後一次的 Commit：
  - `$ git commit --amend`
- 把 Stage 的檔案 Unstage：
  - `$ git reset -- {filename}`
- 把修改過的檔案回復到未修改狀態：
  - `$ git checkout -- {filename}`

19

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

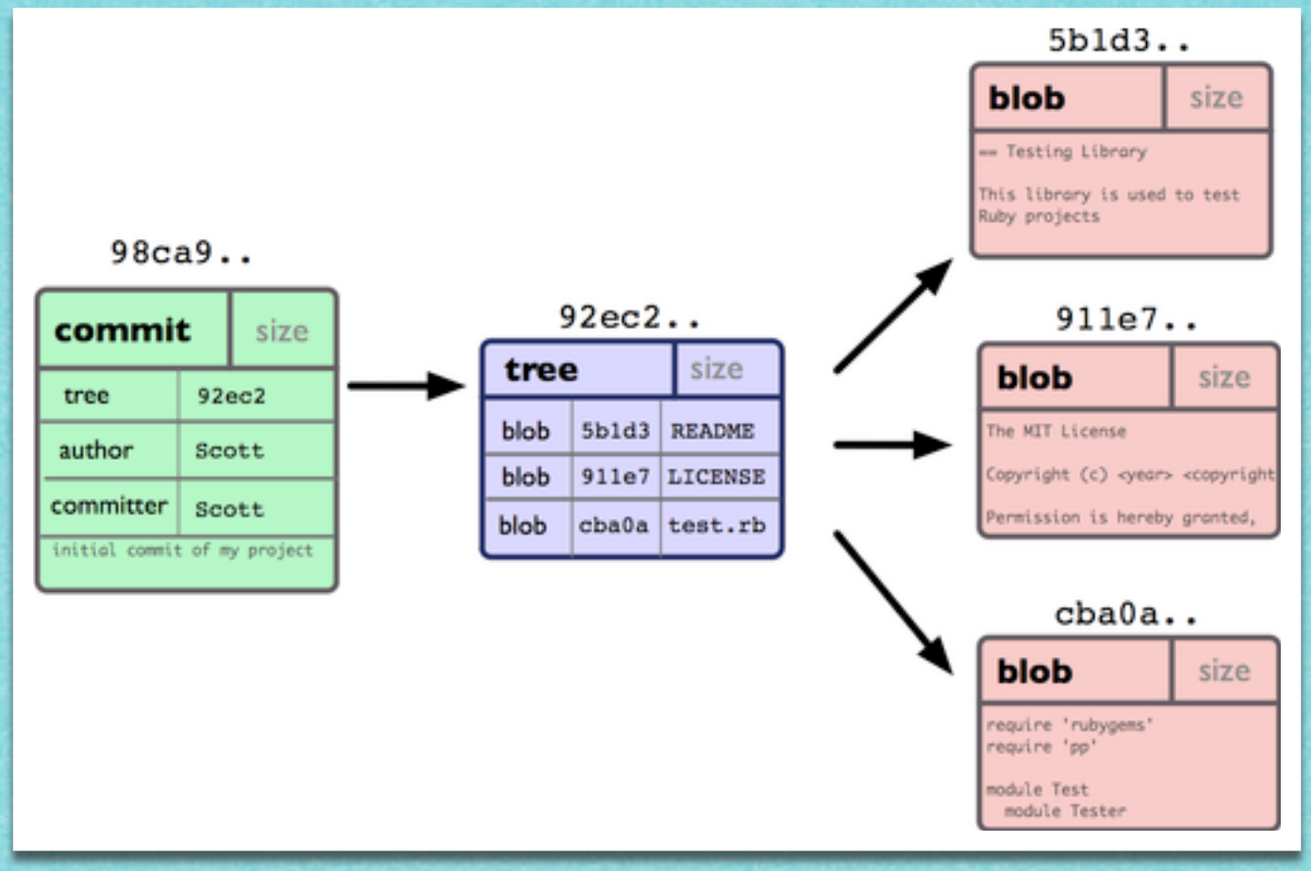
20

"Remote" 的 Protocol

- file://（本機）
- ssh://（效率好、不可匿名訪問、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）

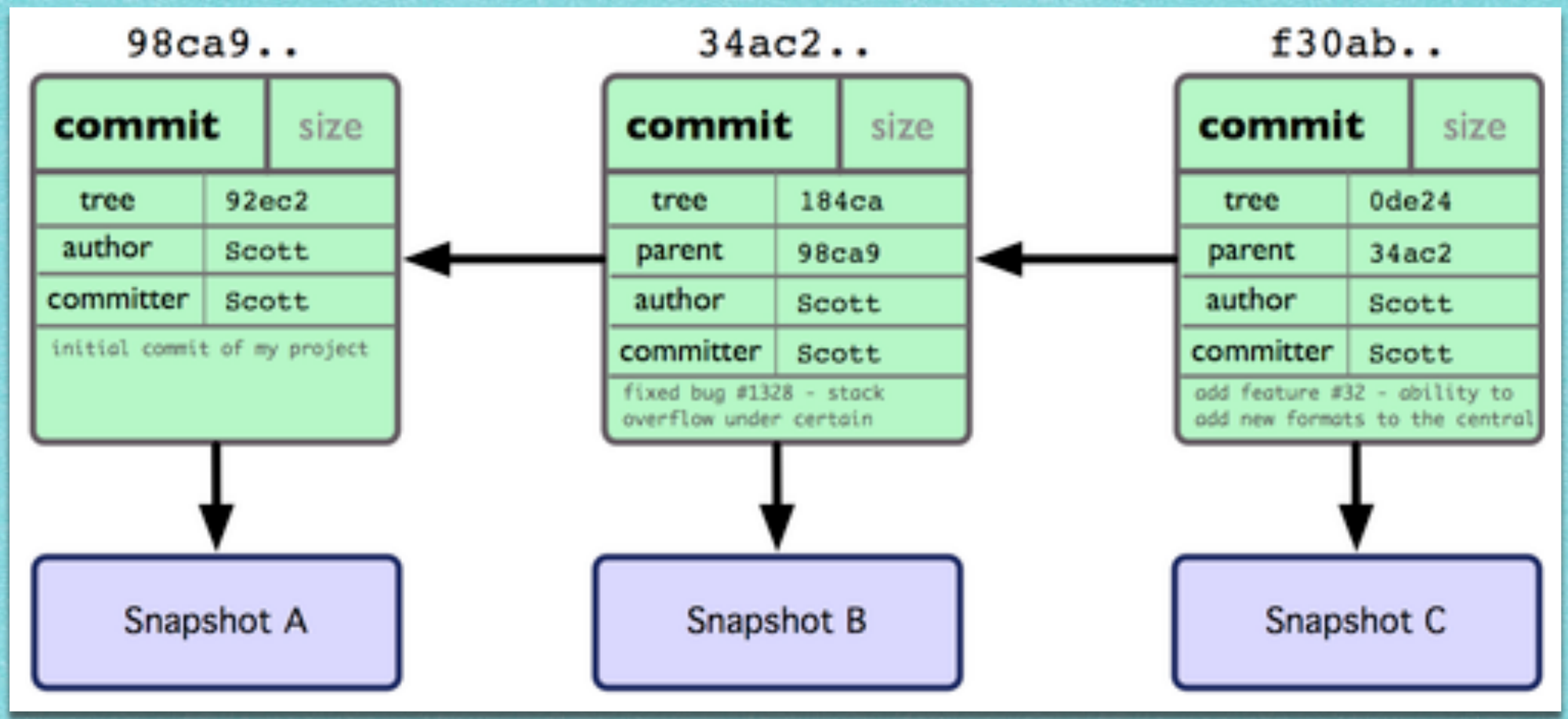
21

Branches



先從 *Commit* 說起

git 是這樣看待 *Commit* 的

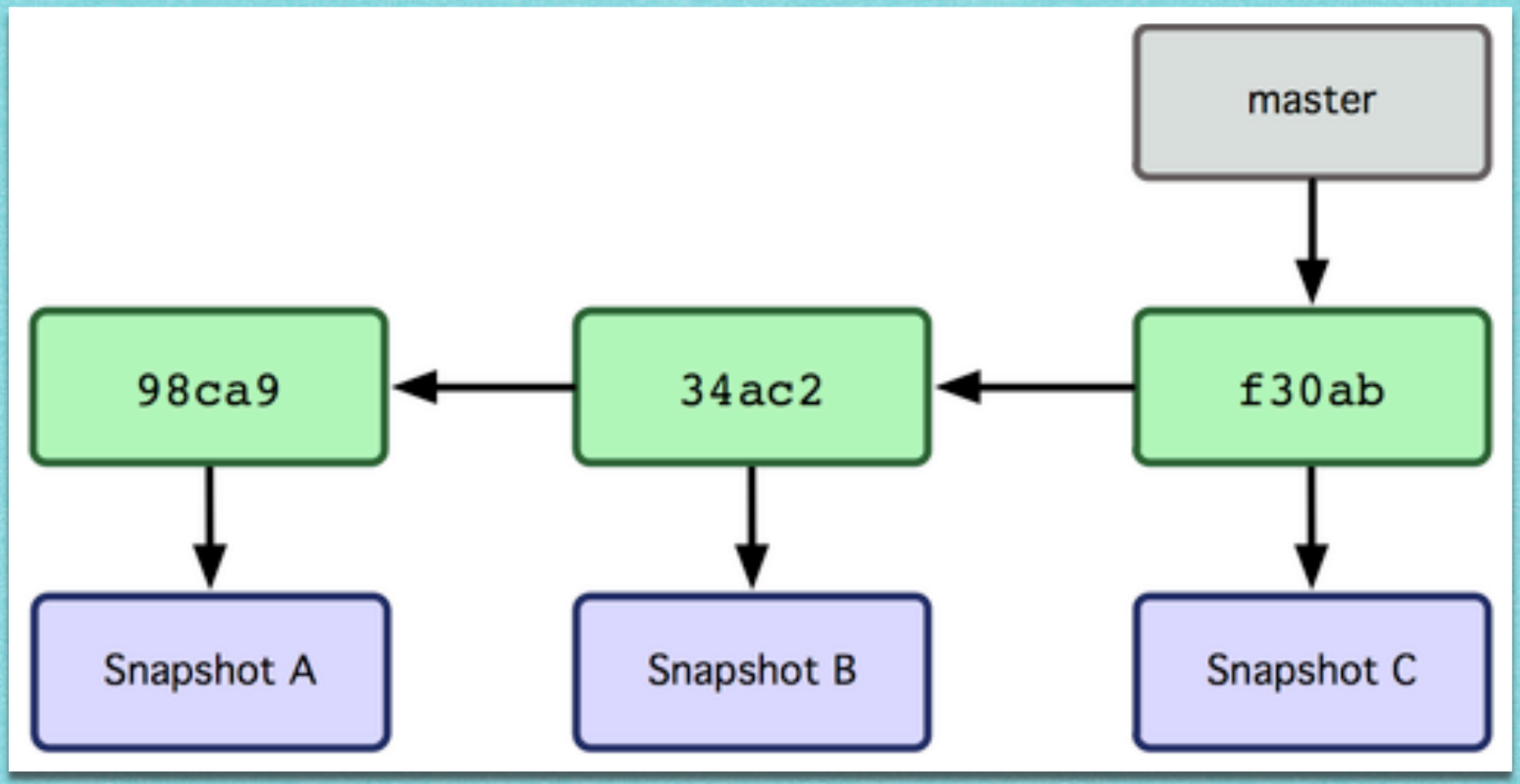


很多 *commit*

*Commit*之間互相連結

每個*Commit*對應一個*Snapshot Tree*

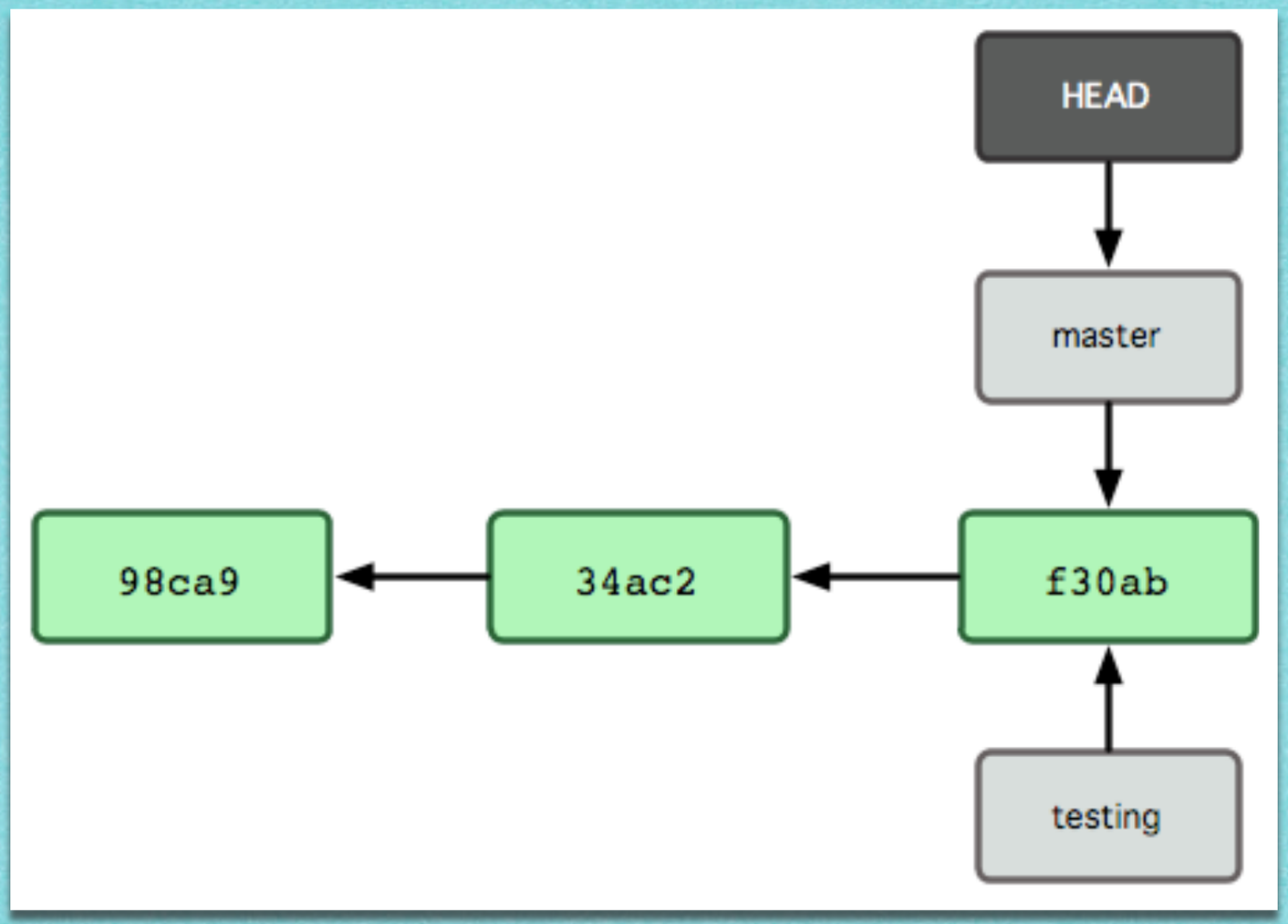
24



於是有了 *branch*

「標記我們在哪個*commit*上」

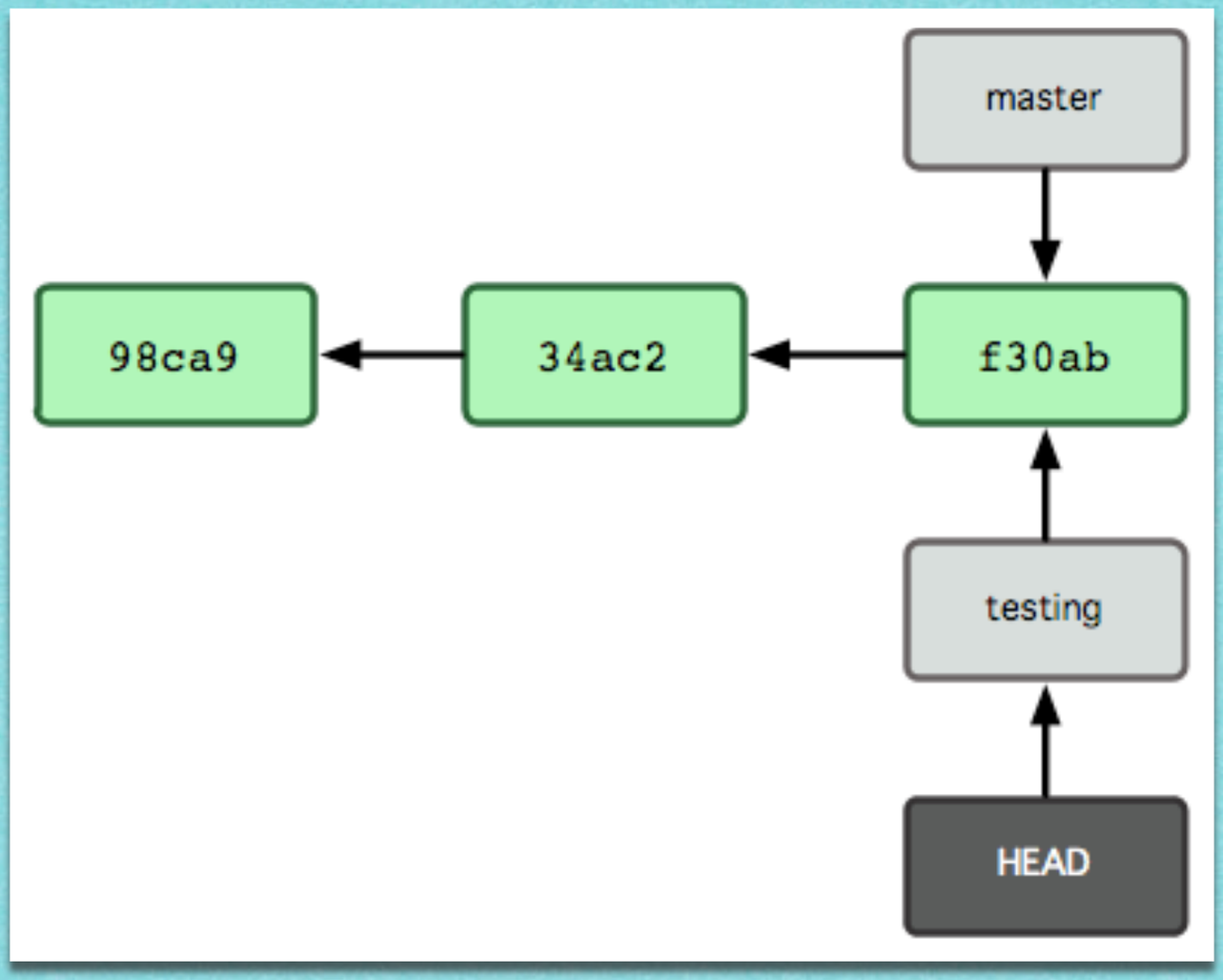
25



$ git branch testing

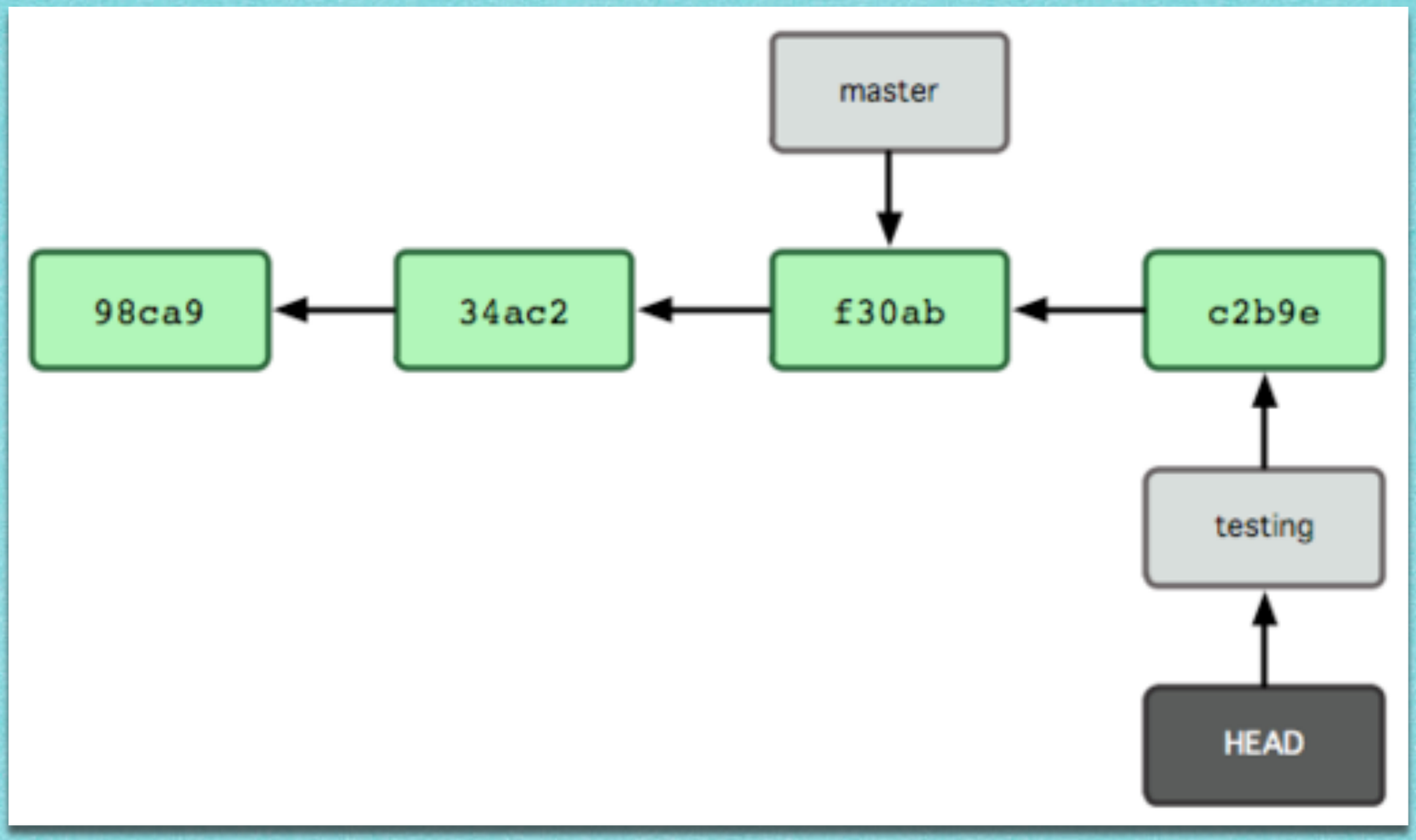
新增一個"*testing*" *branch*

*git*會使用「*HEAD*」紀錄目前所在的*Branch*



$ git checkout testing

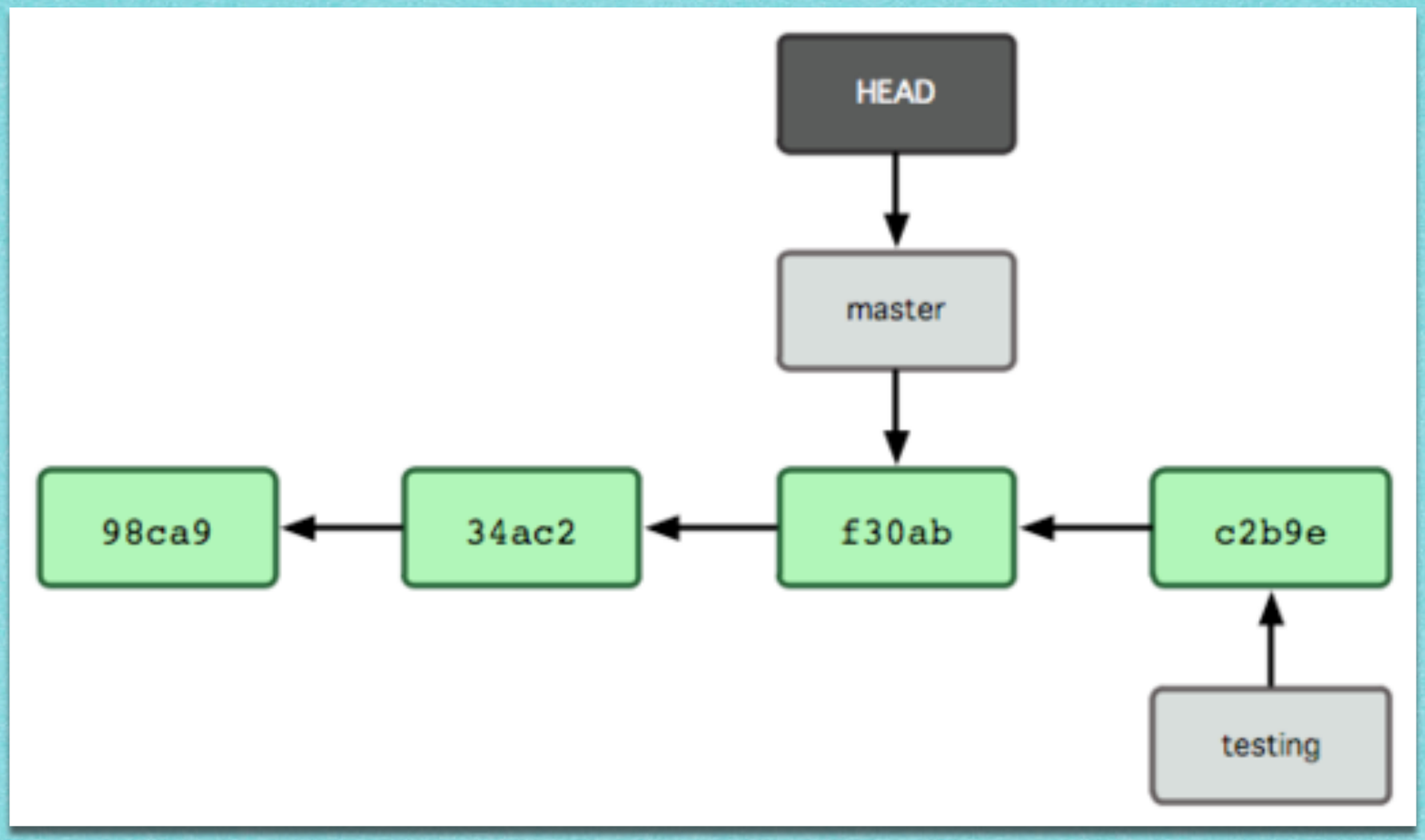
27



這時如果*Commit*下去...

變成只算在 *testing* 不算 *master*

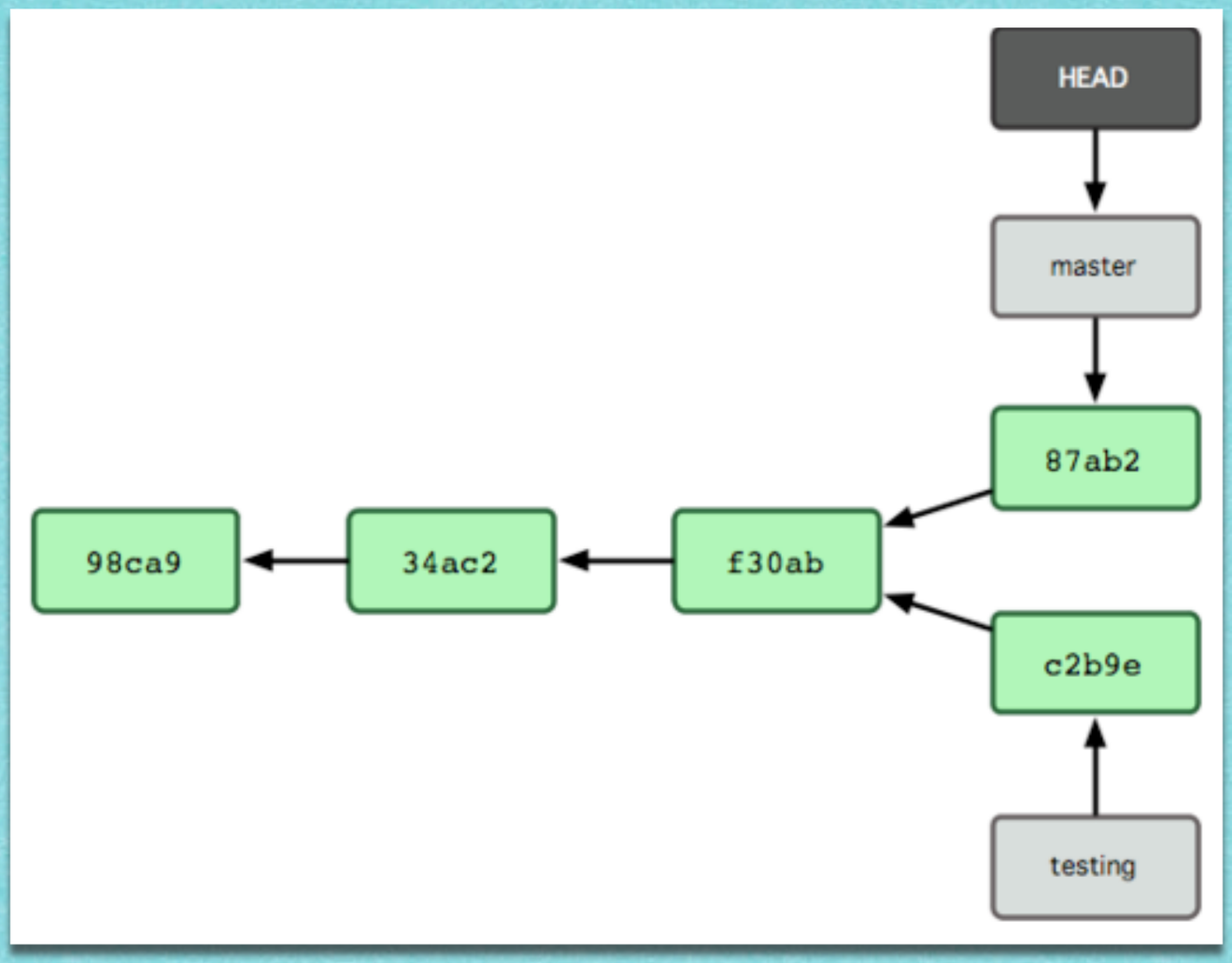
28



$ git checkout master

回到*Master*了，那麼*What's next?*

29




```
$ git commit -m 'made changes'
```

岔開來了！

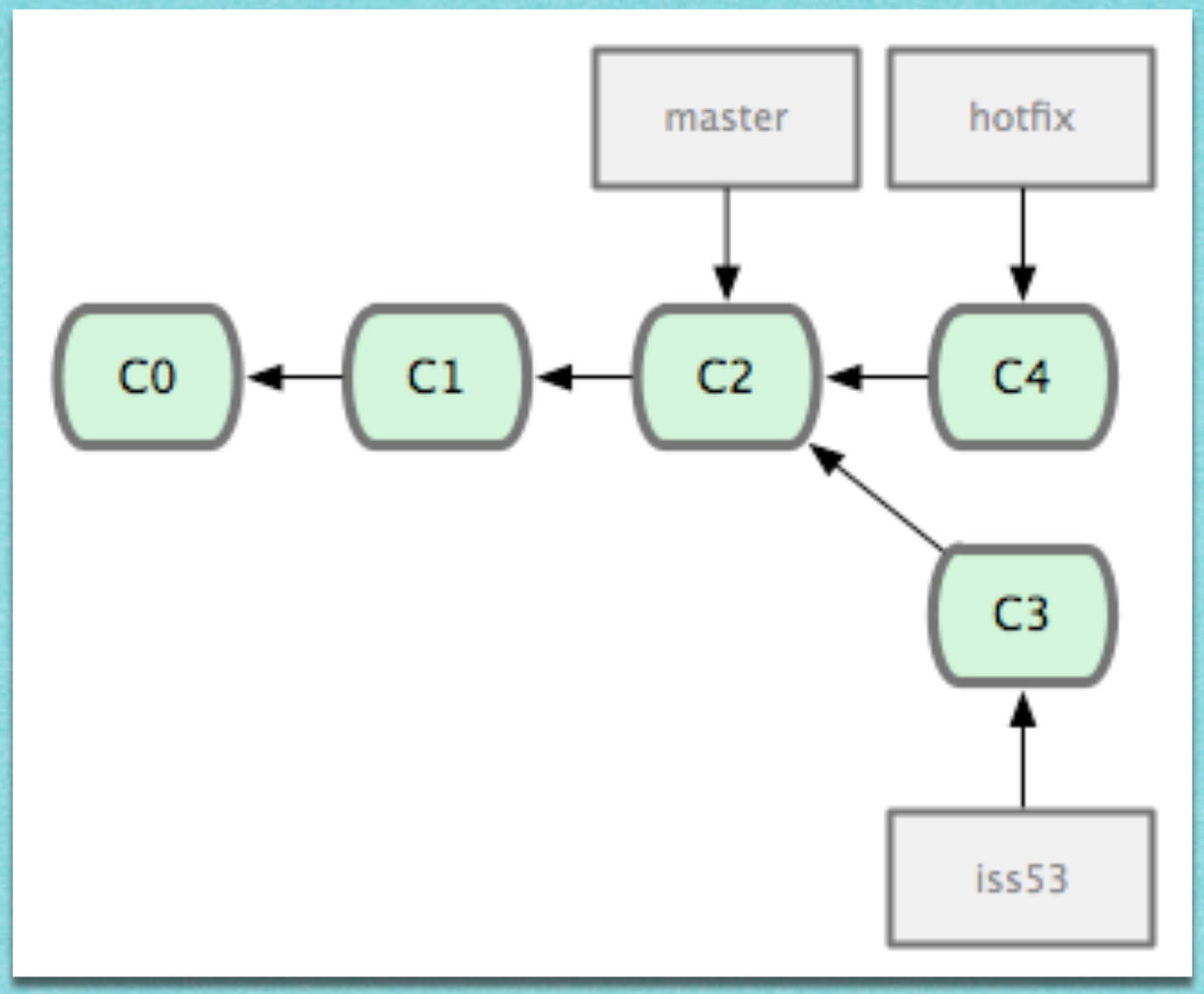
30

如何重新變回一條？

- Merge
  - 通常的建議方式
  - 把岔開來的分支在往後「合起來」
- Rebase
  - 「還沒Push出去的東西」才可以Rebase！
  - 把岔開來的分支「裝回去」主線

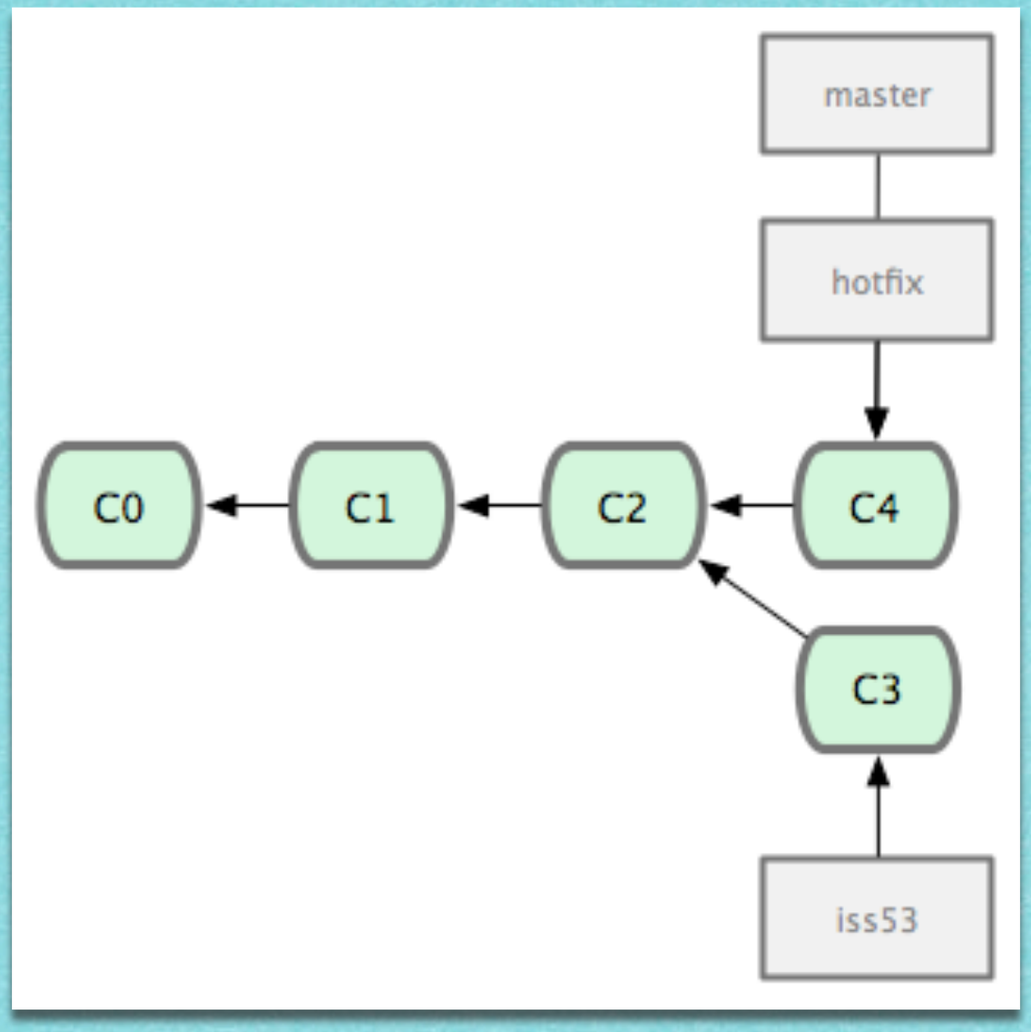
31

Merge



*Branch*滿天飛

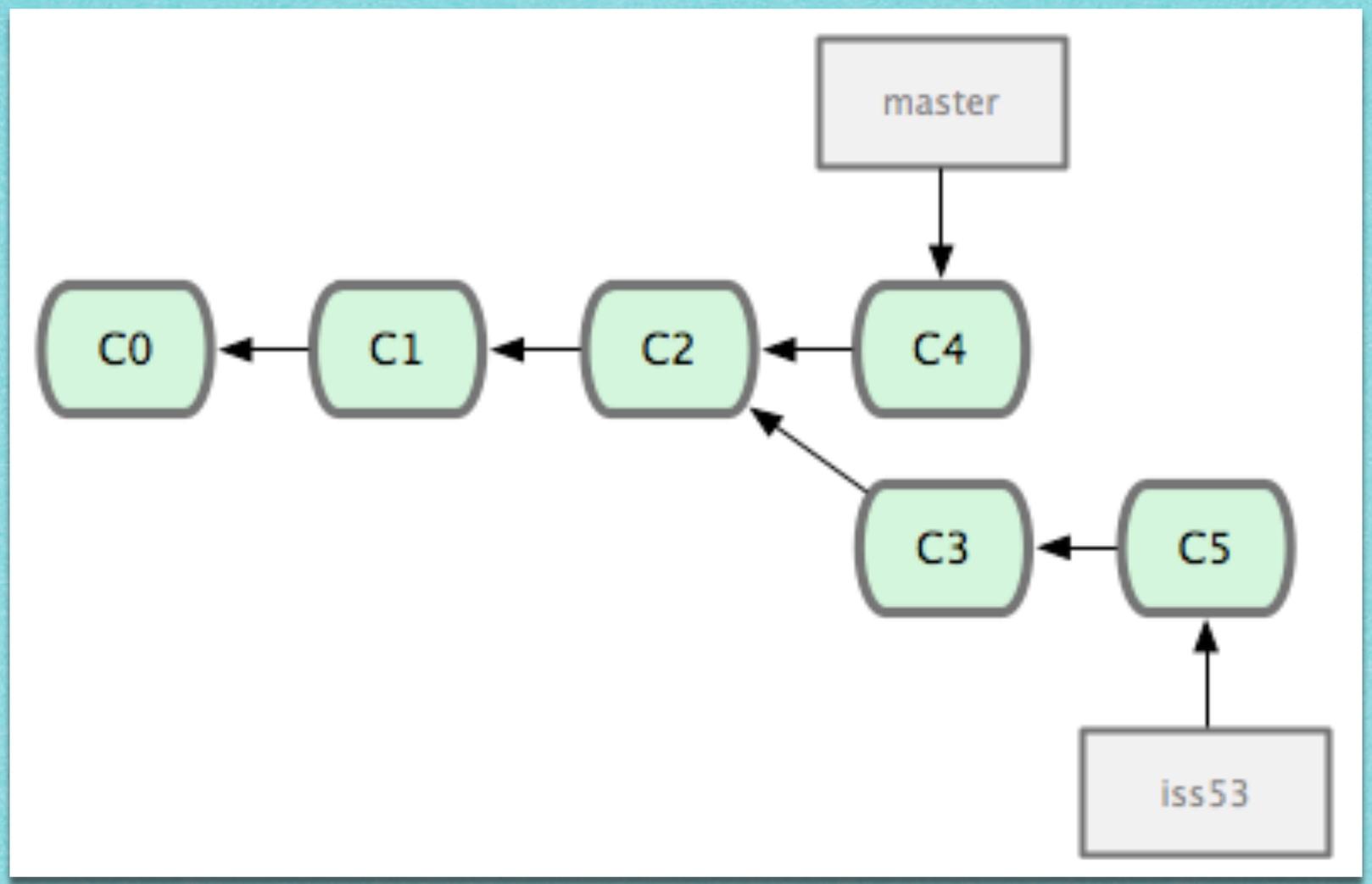
有緊急的 *Bug*，我開了一個 *hotfix branch* 弄好了。
現在要怎麼處理 *master*？




```
$ 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*

34



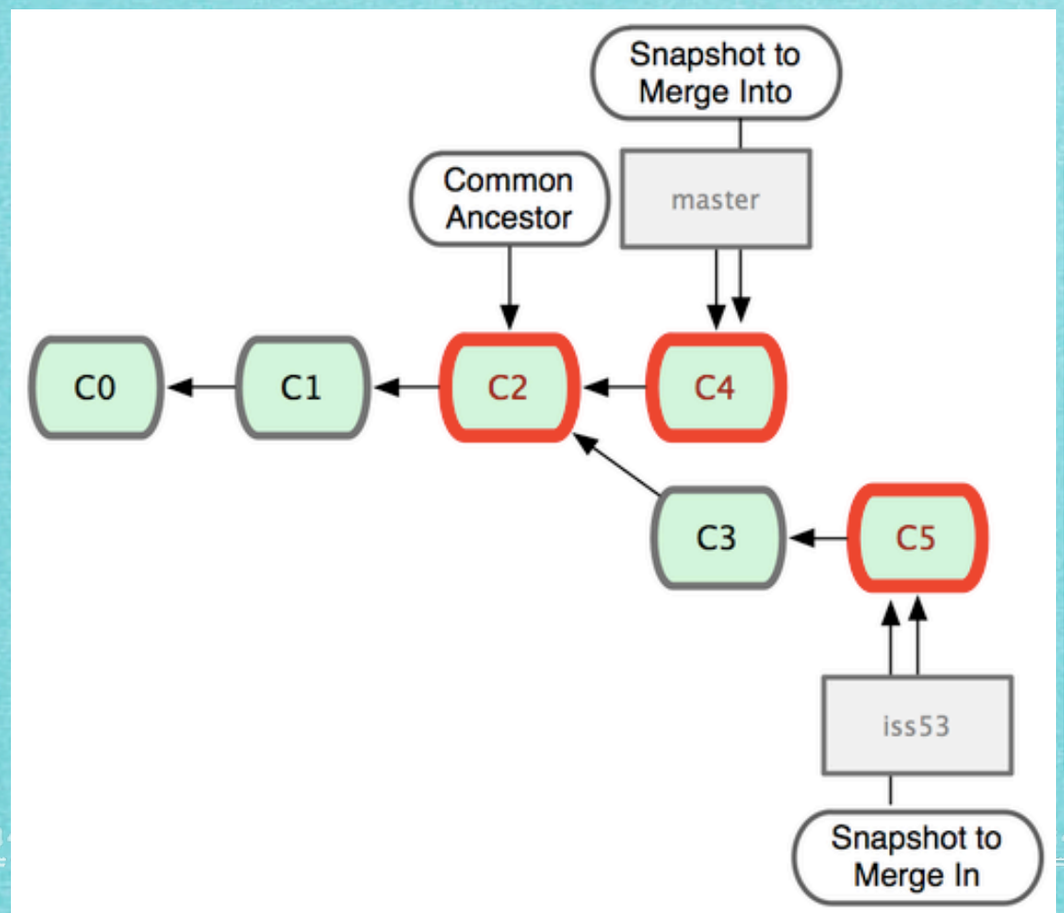
*Branch*滿天飛

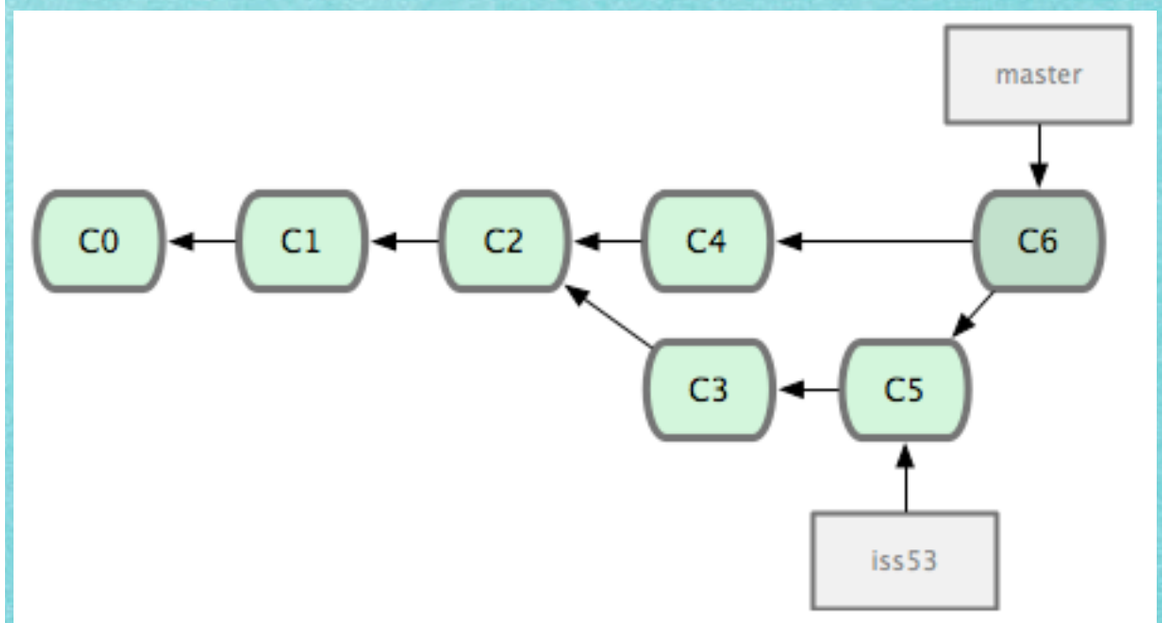
好，我繼續來寫 *Issue 53* 的 *branch*！
總算寫完了！我要*Merge*！

35

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(-)
```





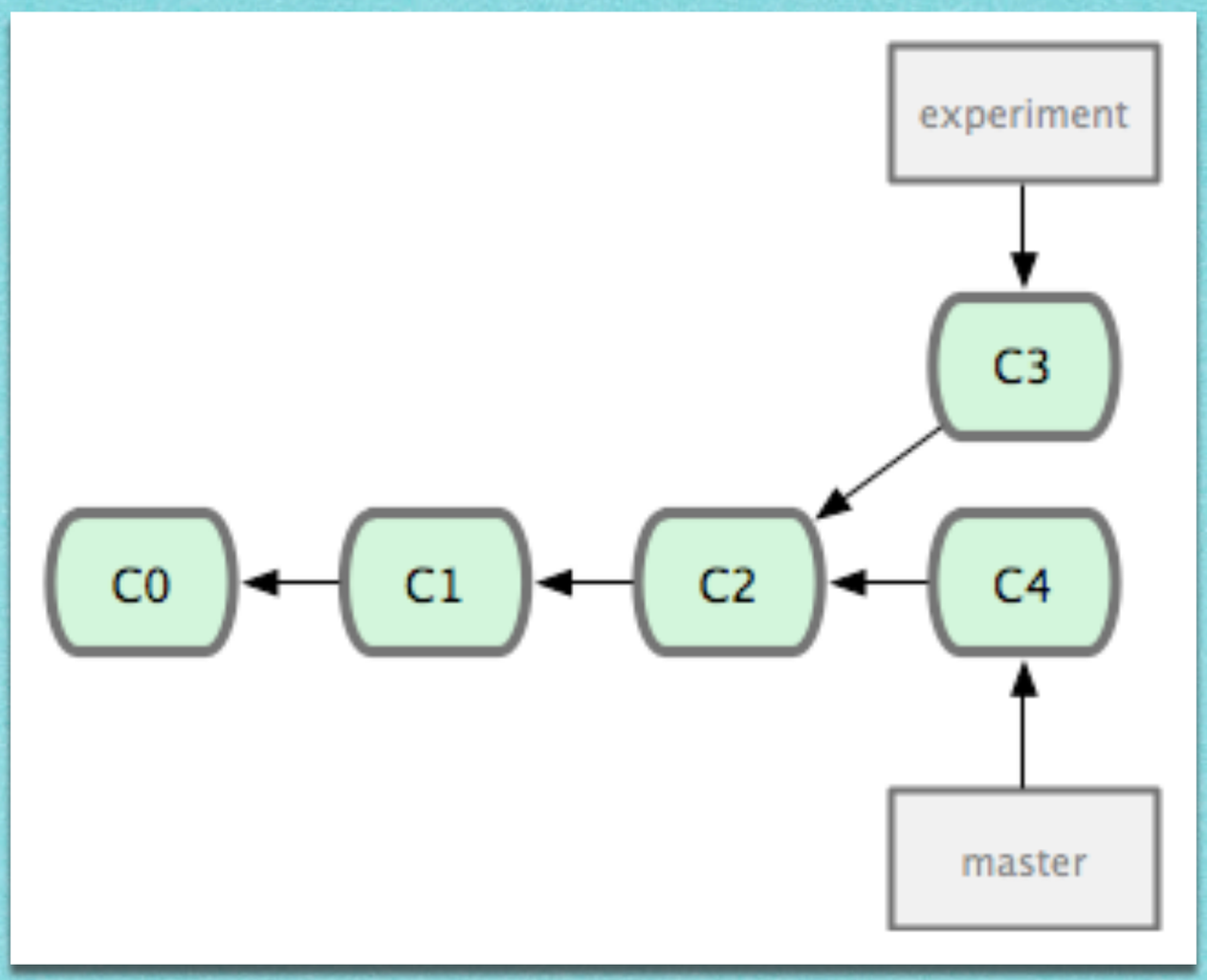
Conflict！衝到了！

- 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
```

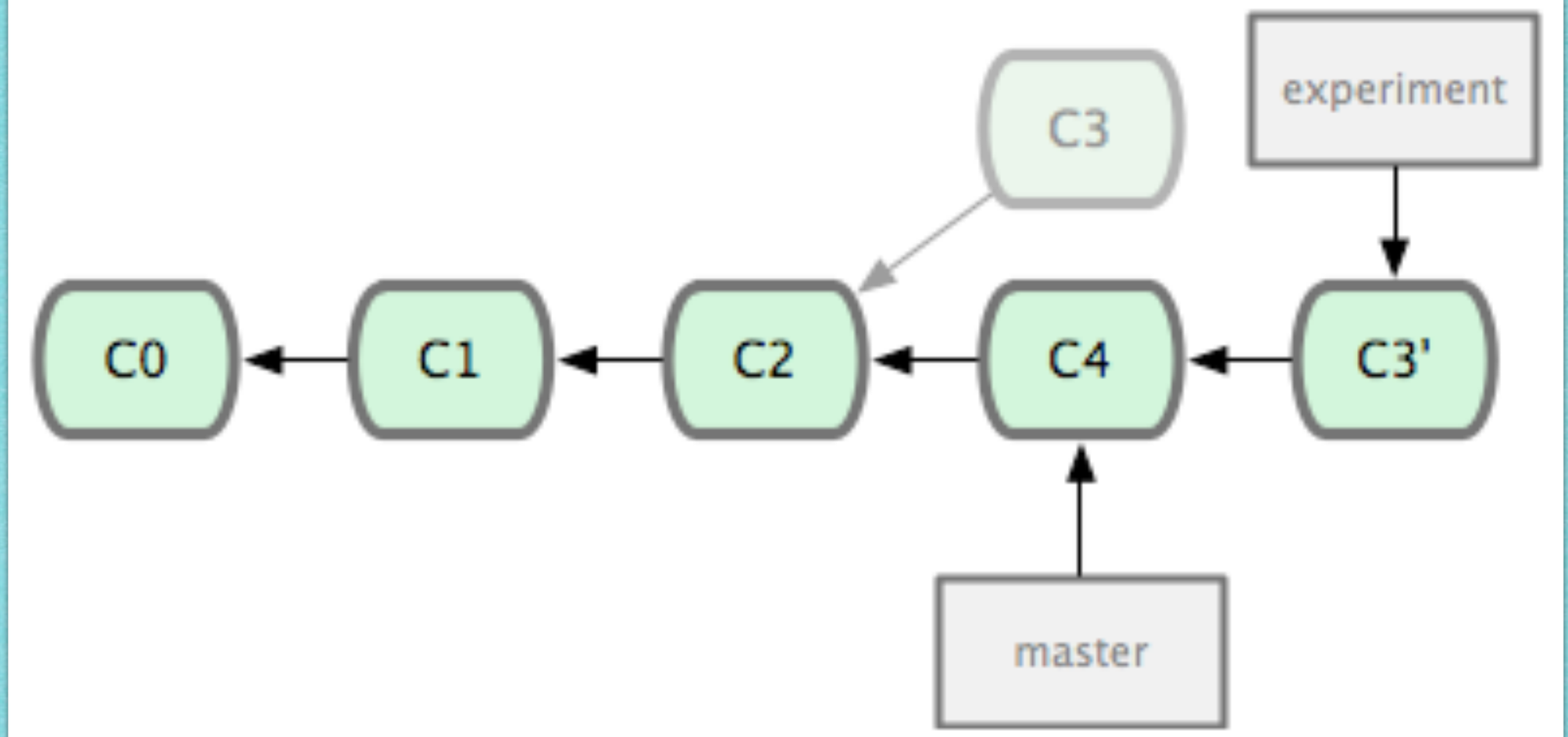

37

Rebase



回到類似先前的情境

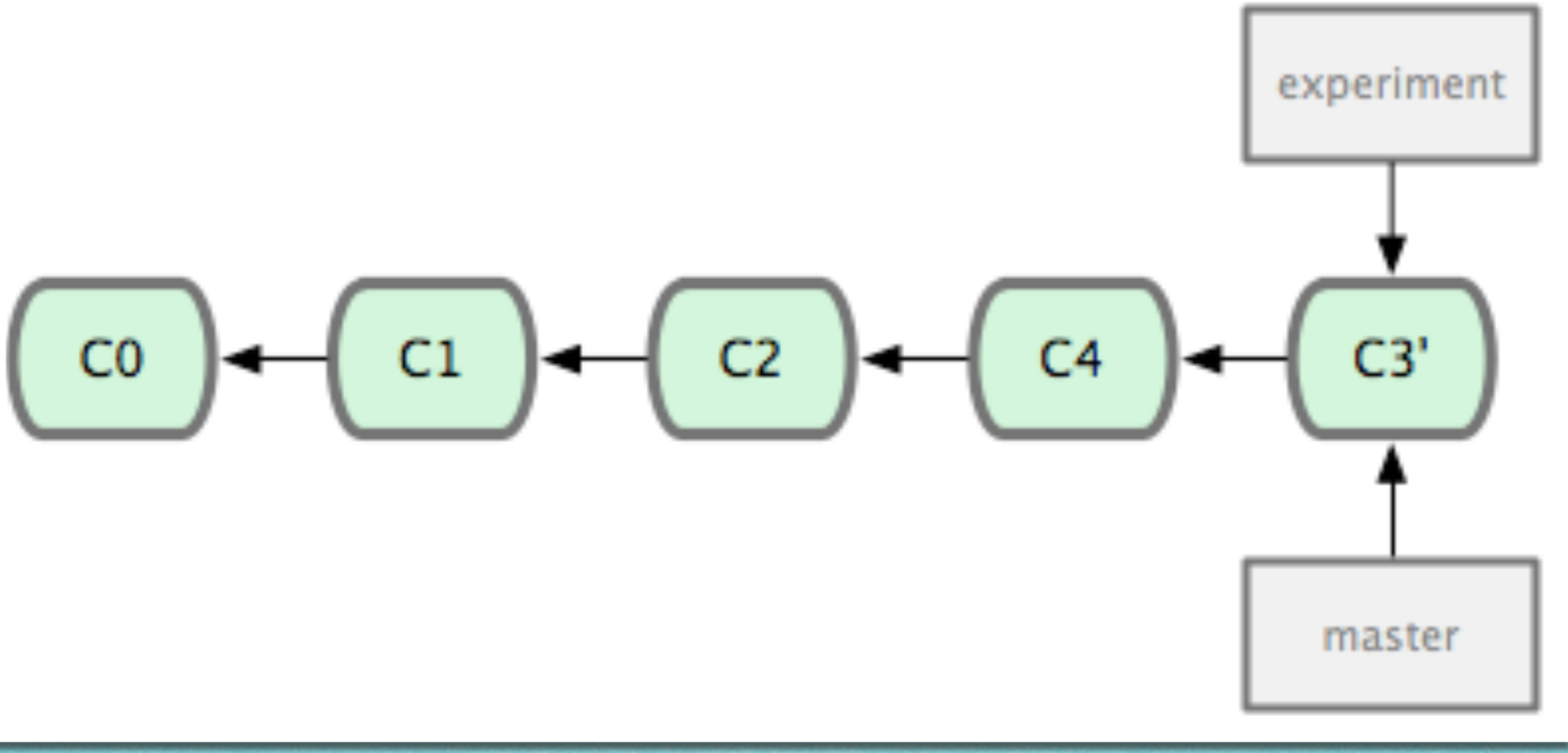
除了*Merge*以外，我有沒有其他的辦法可以處理掉 *experiment branch?*




```
$ git checkout experiment
$ git rebase master
```

把*C3*上的變更*Rebase*到*C4*之上*(**onto** C4)*

40

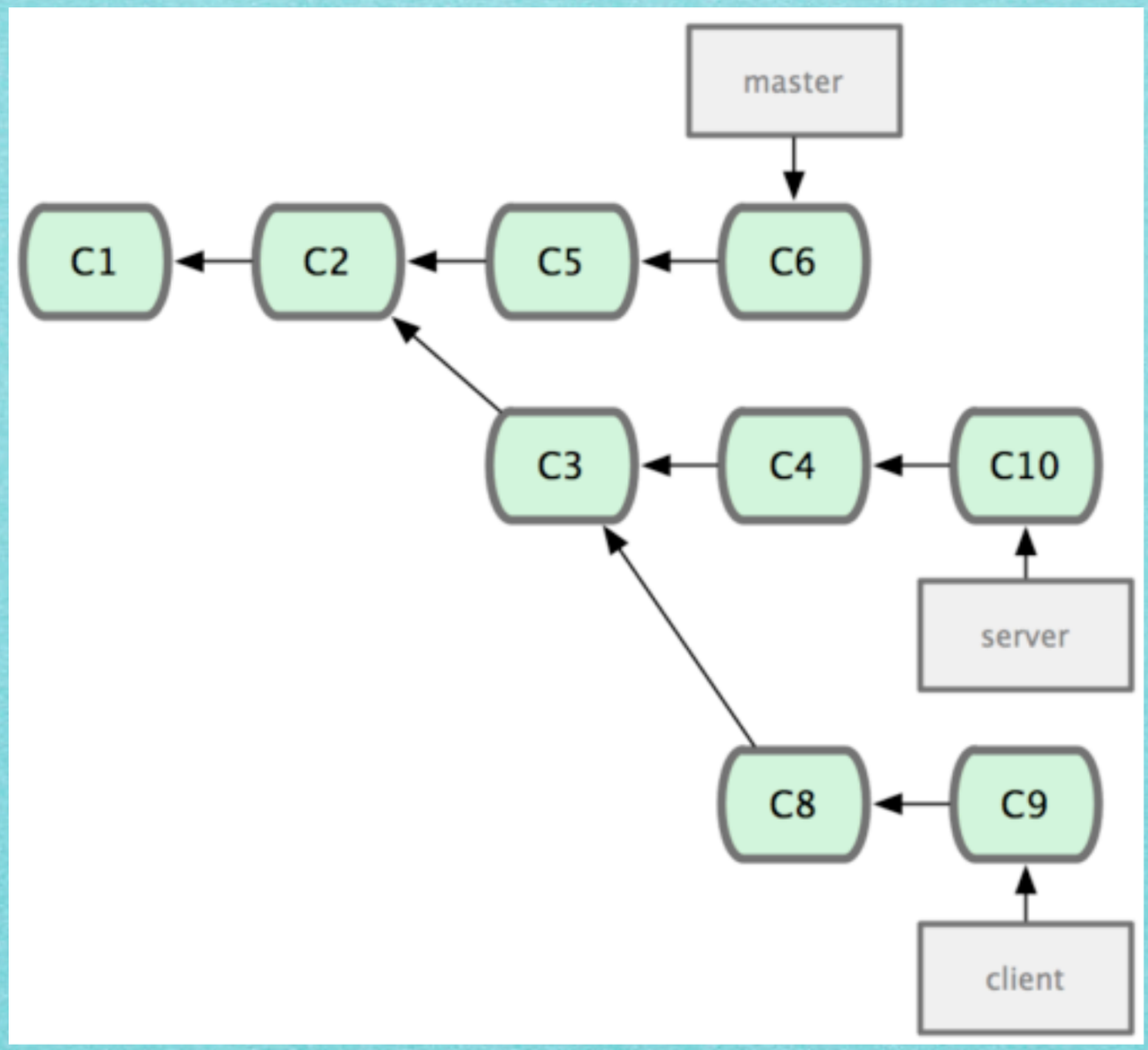



```
$ git checkout master
$ git merge experiment
```

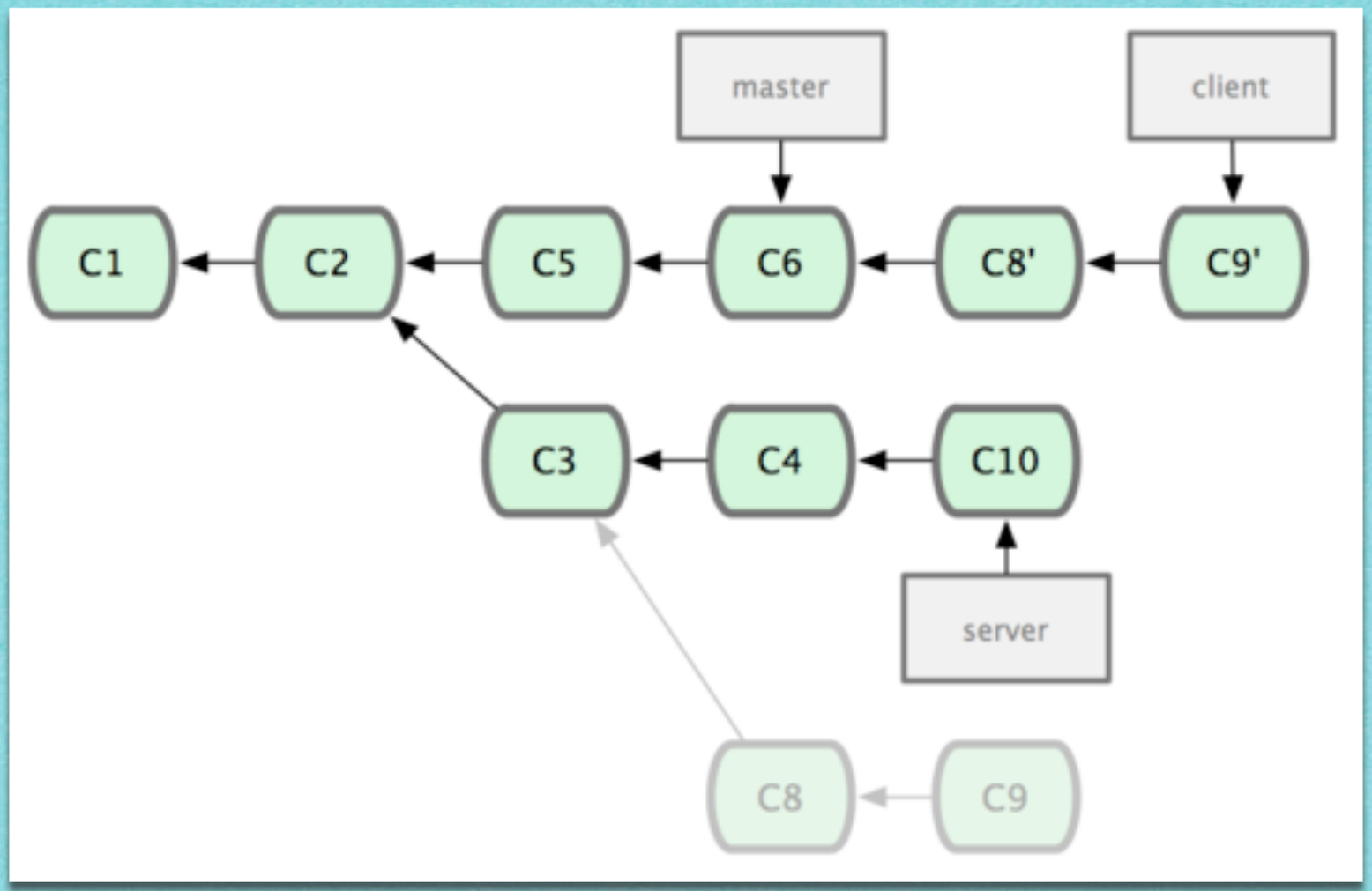
Rebase過後就可以fast-forward Merge了！

41

當事情更複雜...



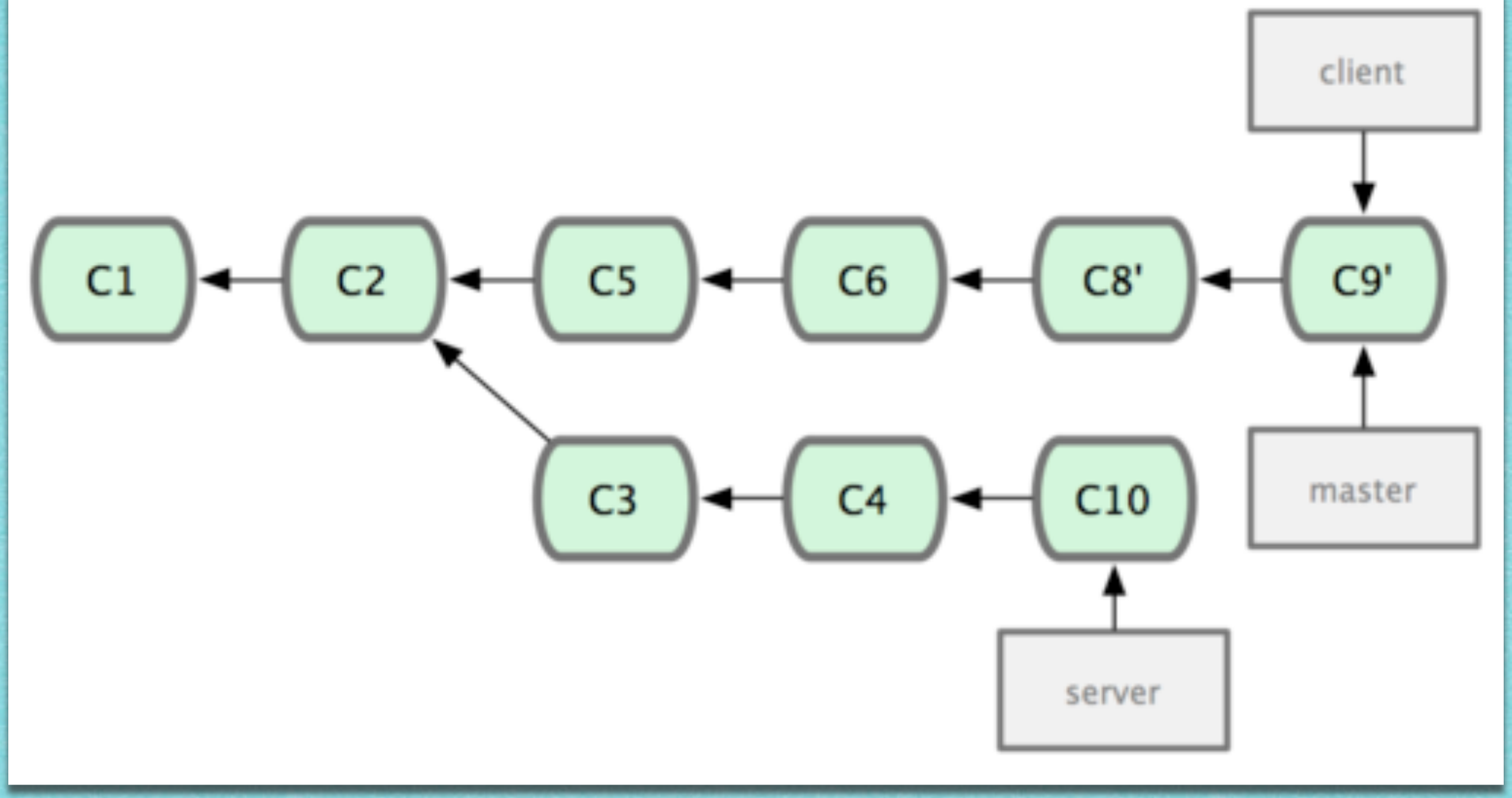
42




```
$ git rebase --onto master server client
```

「檢查*client*的*Branch*，從*client*和*server*的共同祖先看作了哪些變動，把那些變更重新在*master*上面作過一遍」

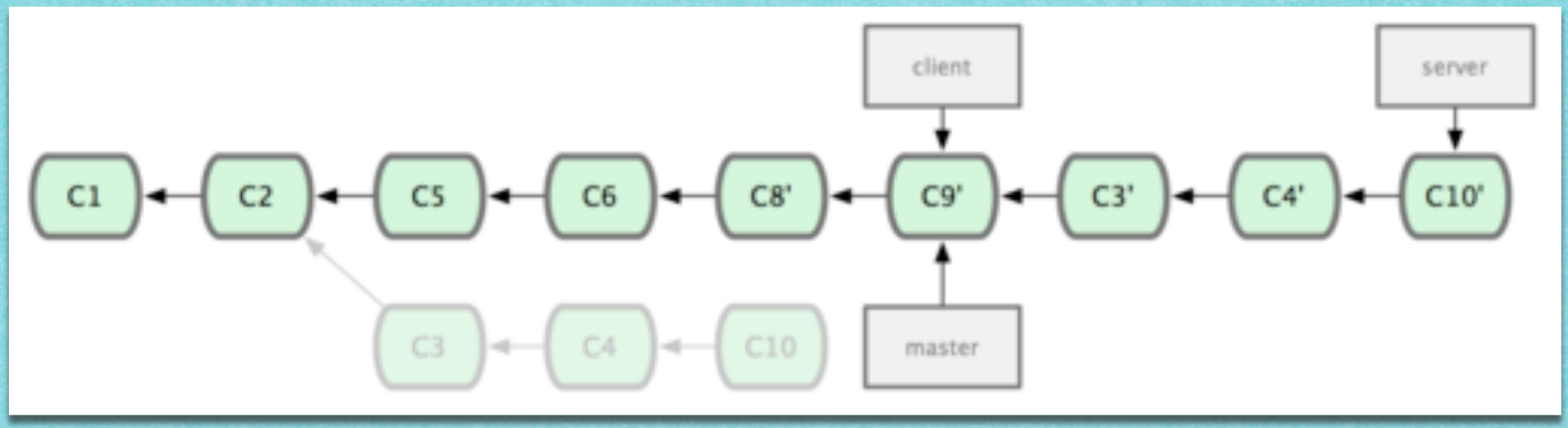
43




```
$ git checkout master
$ git merge client
```

對*client*作*fast-forward Merge*

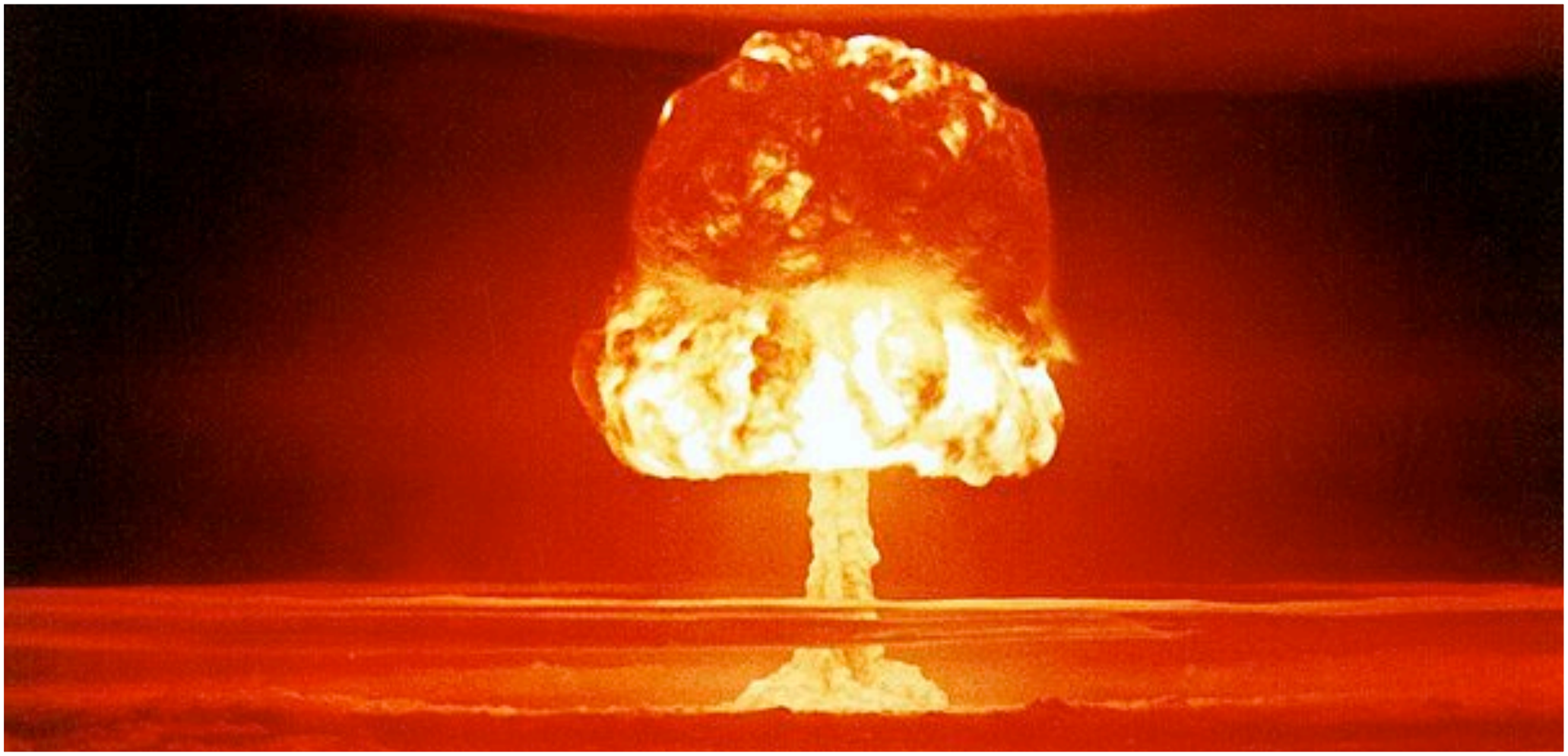
44




```
$ git rebase master server
$ git checkout master
$ git merge server
$ git branch -d client
$ git branch -d 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*）
通通都*Merge*完之後，把兩條不要的*Branch*砍掉囉 *:)*

45



「千萬不要」對已經*Push*的東西作*Rebase*！

除非你想成為害群之馬。

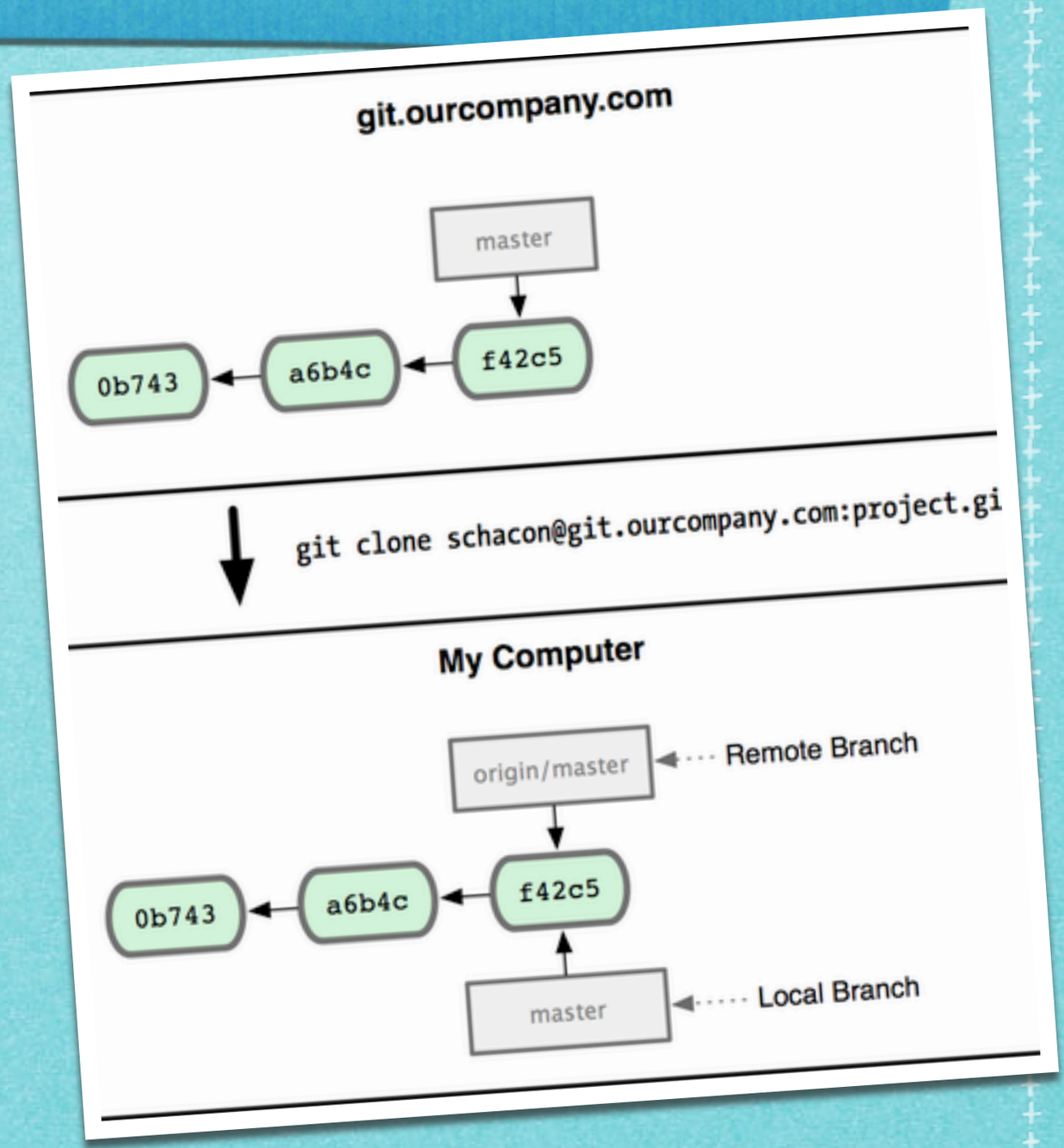
Pic: http://commons.wikimedia.org/wiki/File:Castle_Romeo.jpg, Public Domain, United States Department of Energy

46

Remote Branches

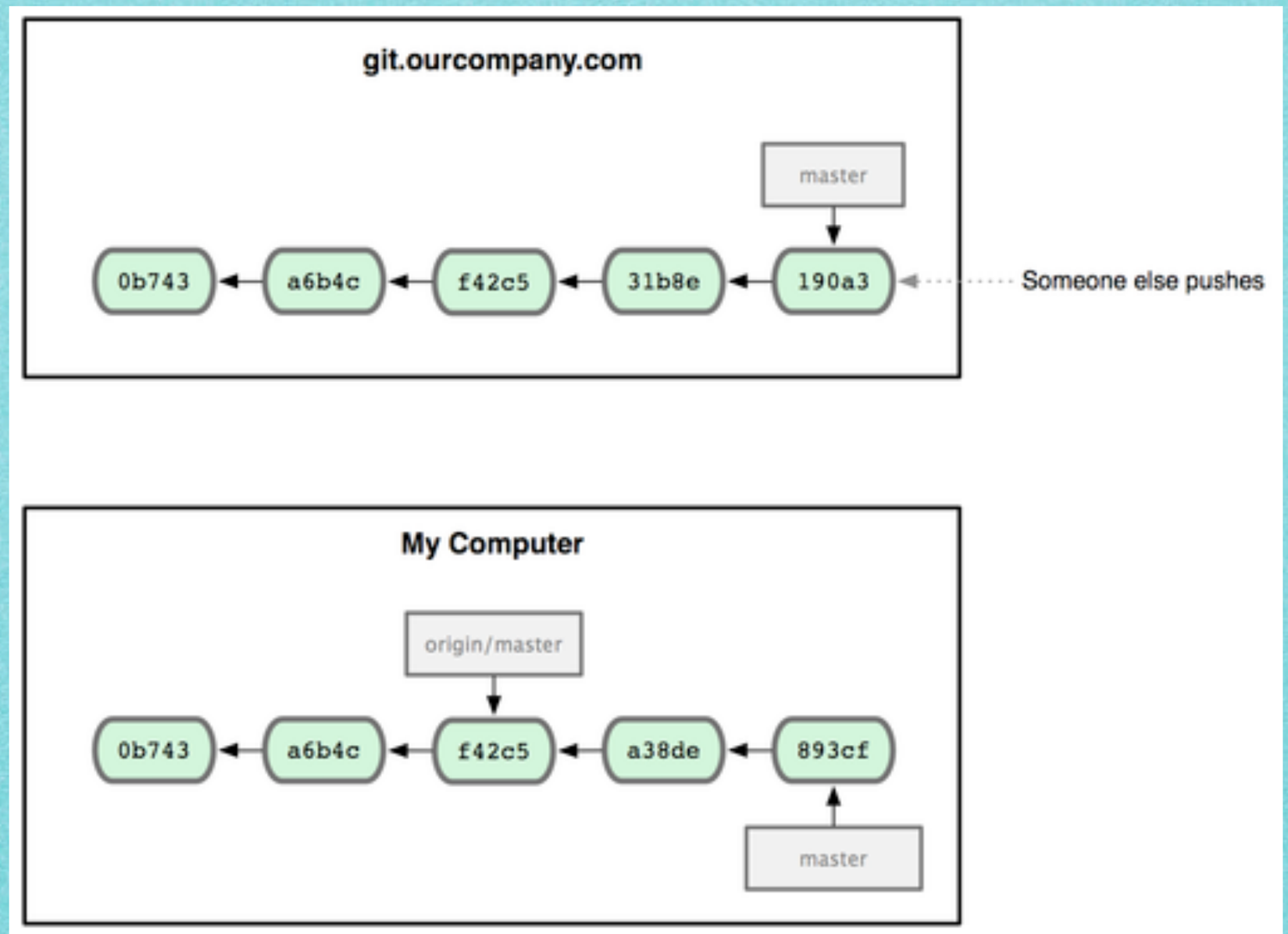
Remote Branch

- 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



48

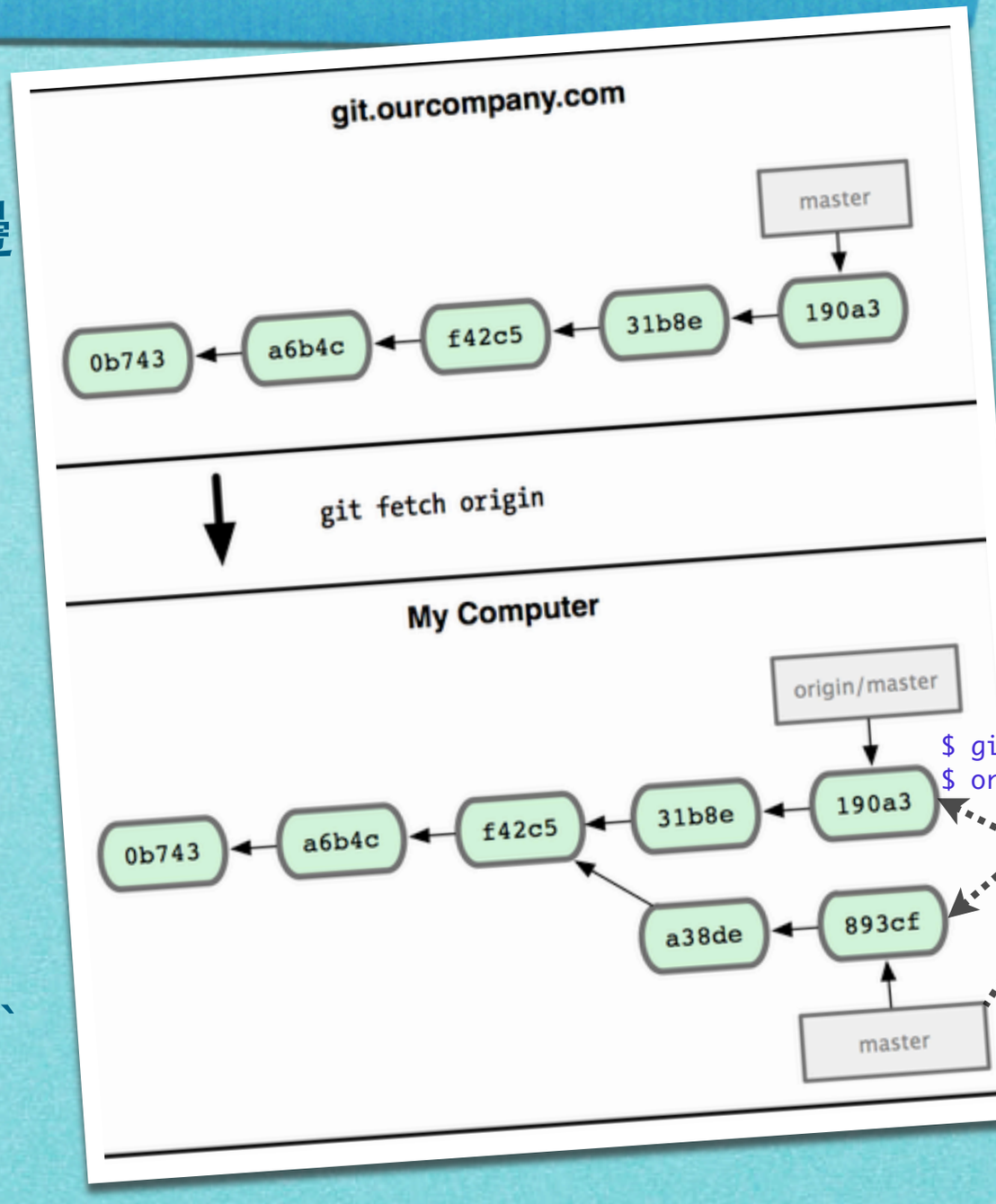
當你與Remote Branch 同時改了東西...



49

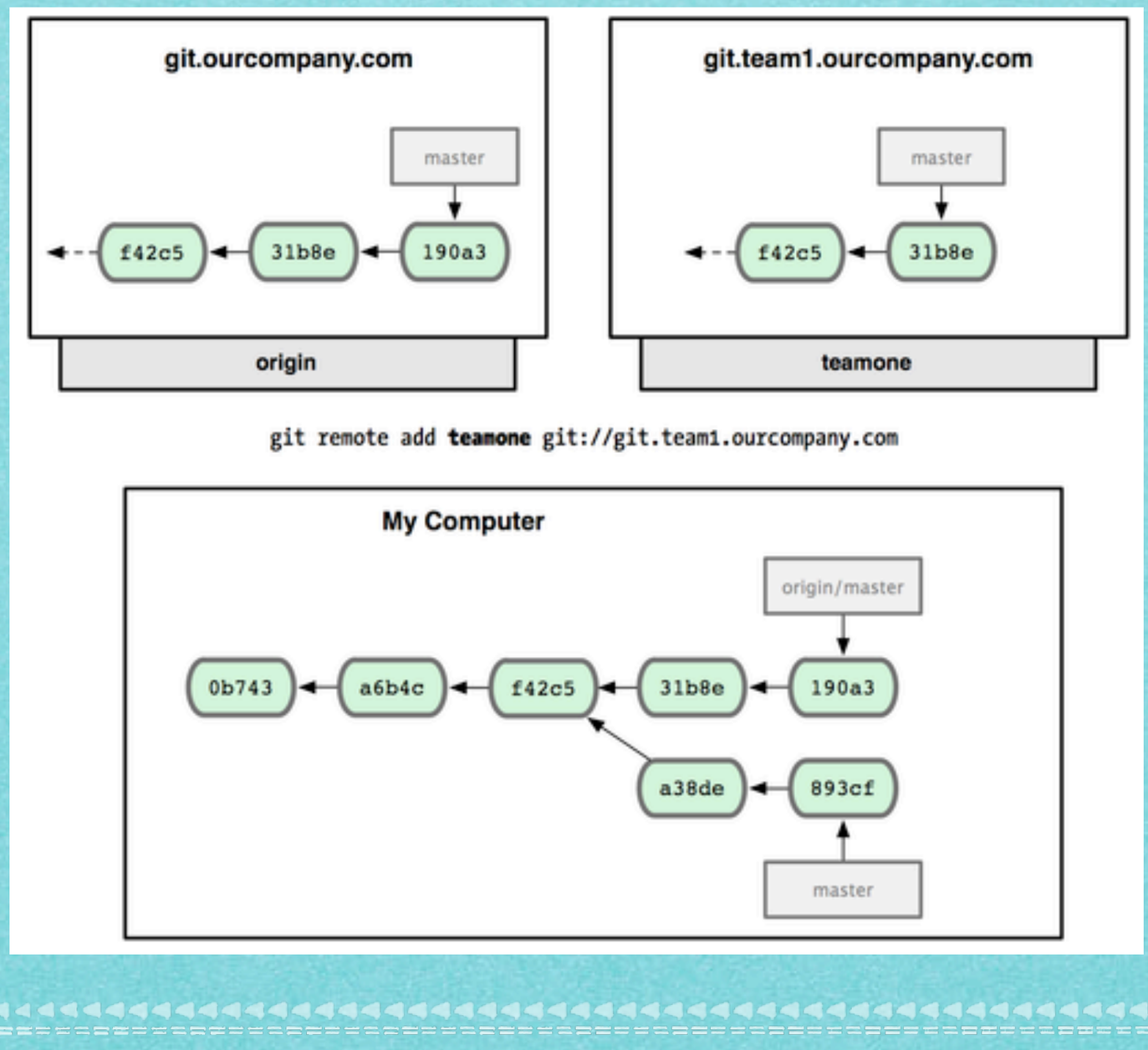
Pull = Fetch + Merge

- 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`+
    `git merge origin/master`



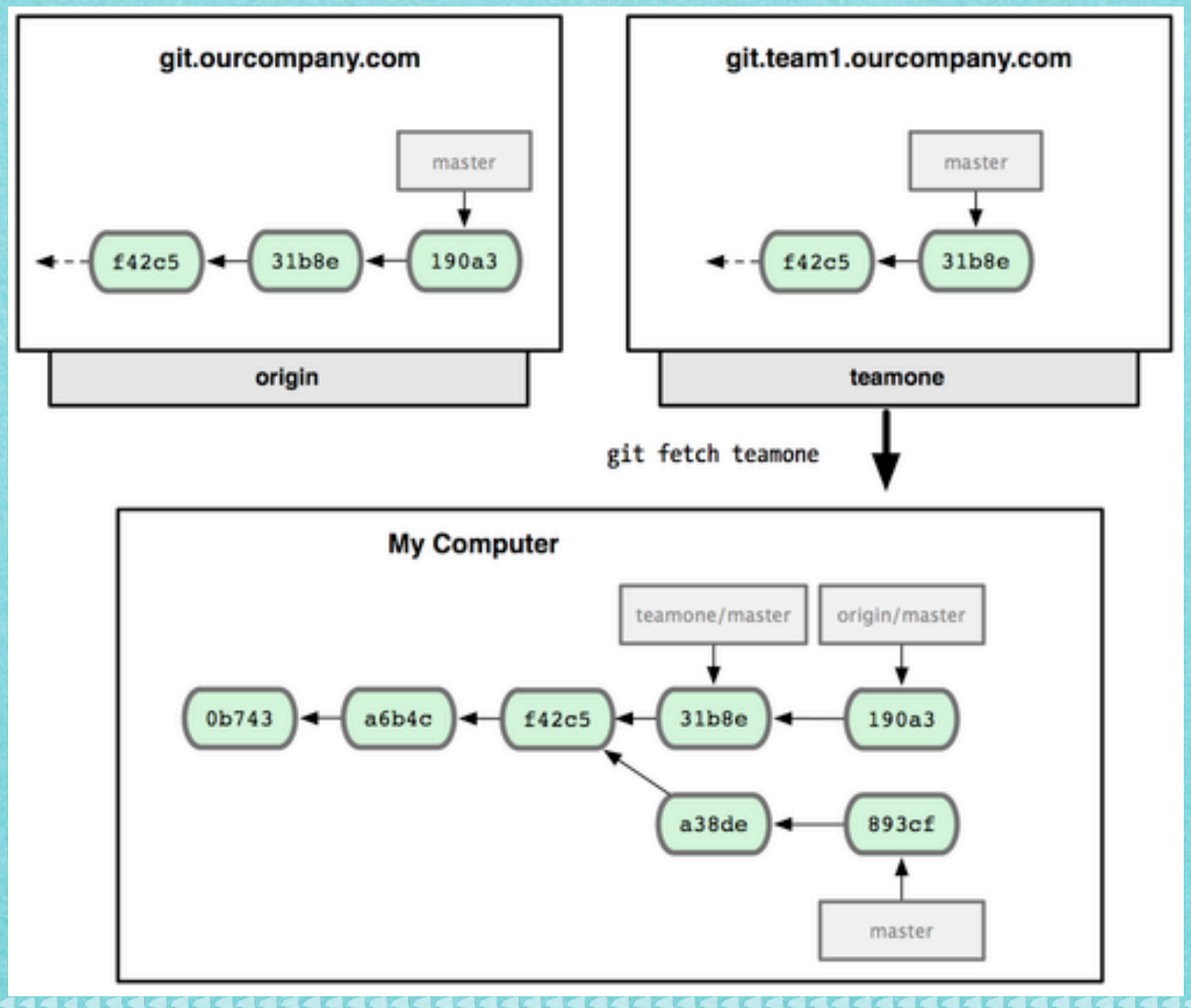
50

新增一個Remote



51

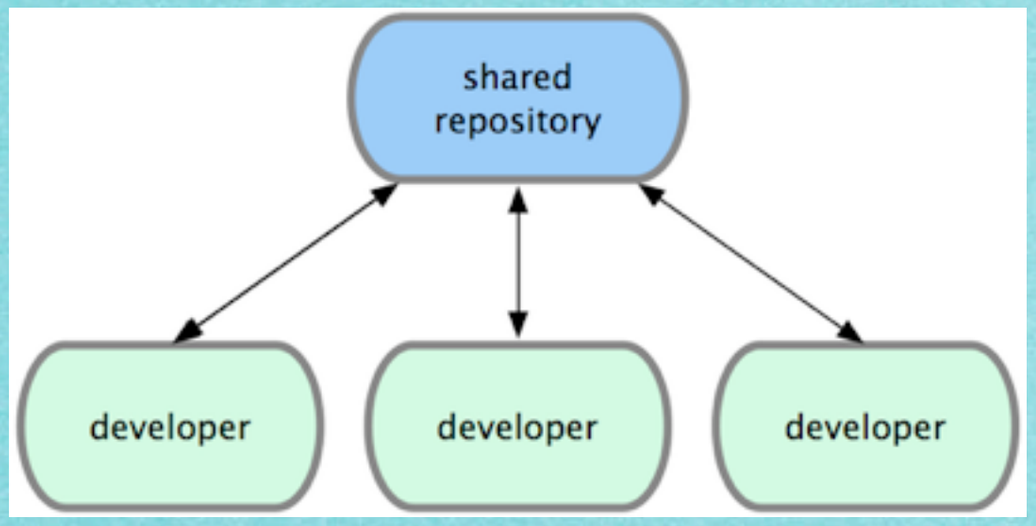
Fetch後會有新的Remote Branch



52

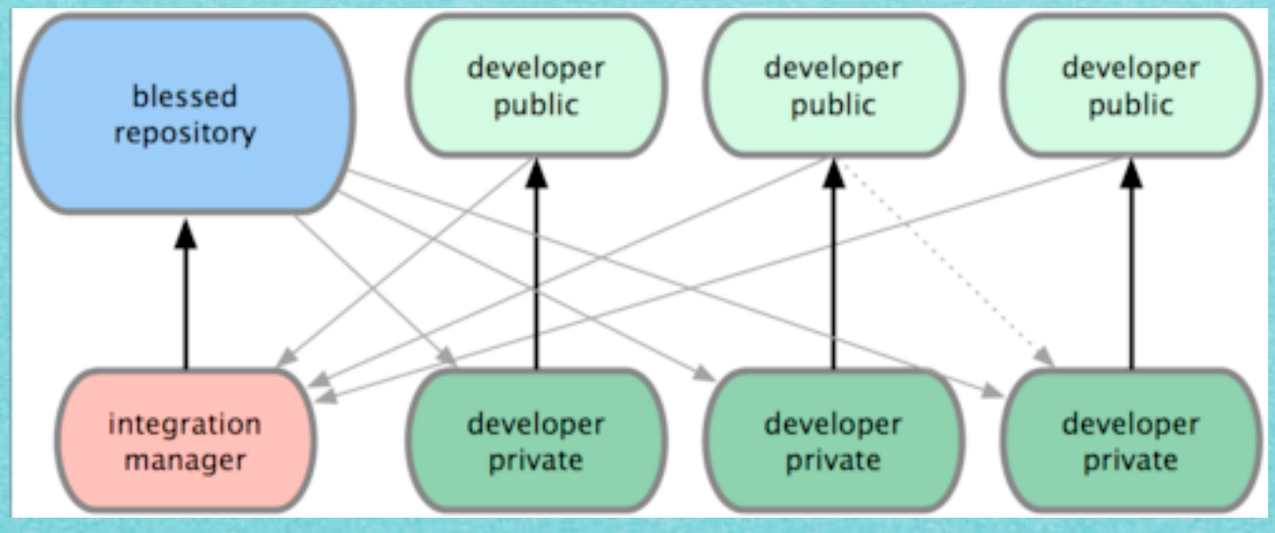
Distributed Workflows

開發方式任你選擇

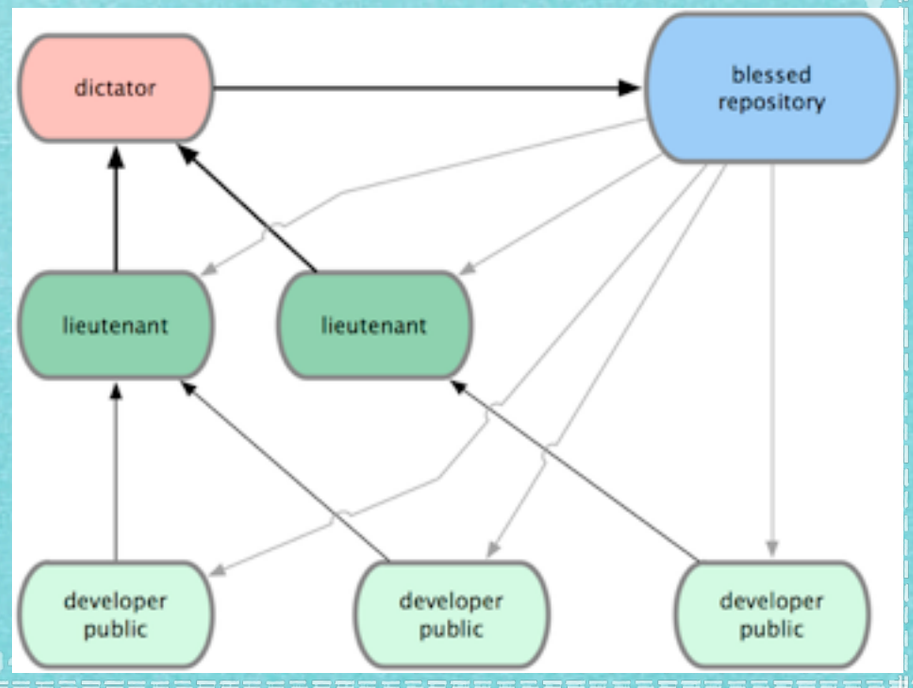


中央式的開發

整合管理員



司令官和副手



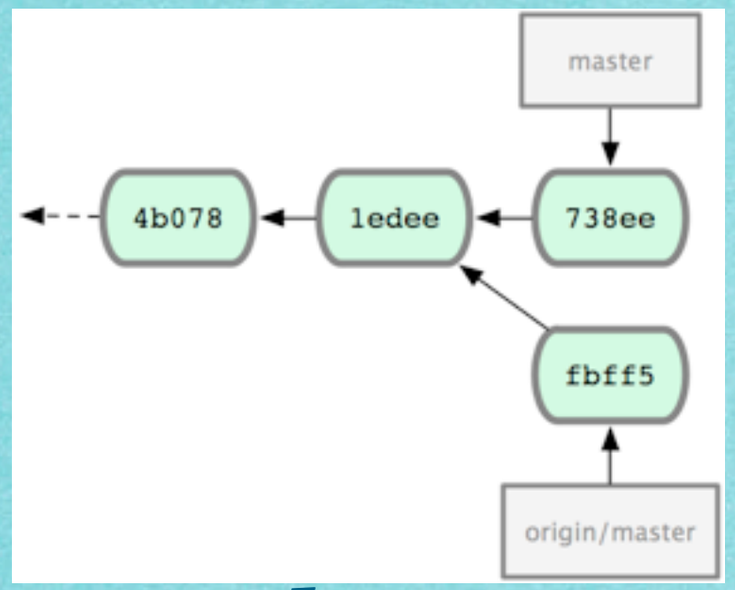
54

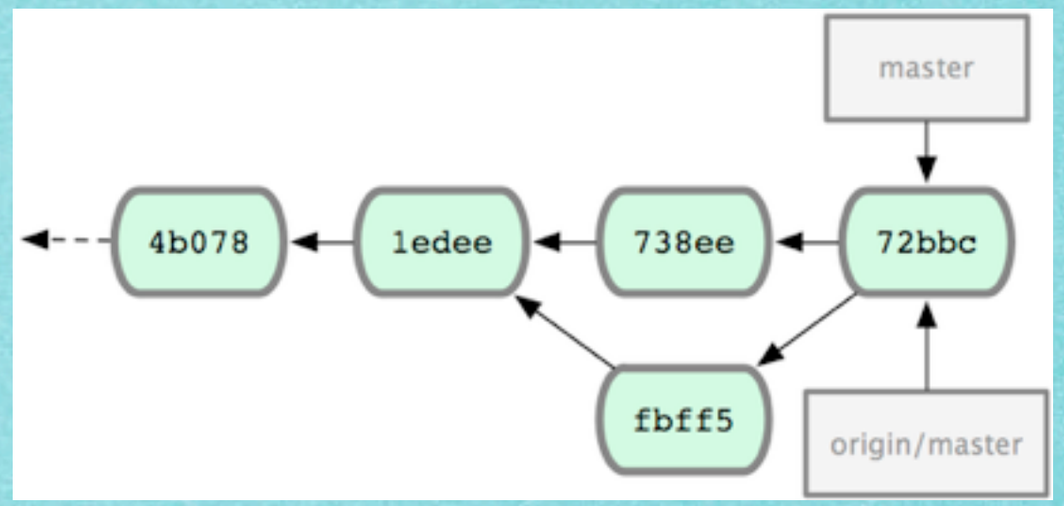
一些基本守則

- 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！

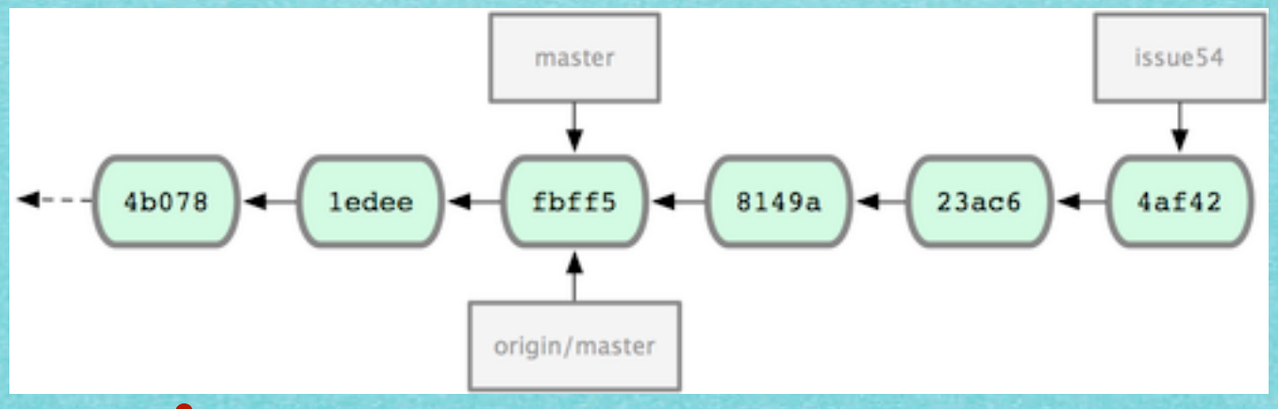
55

私人小團體協作





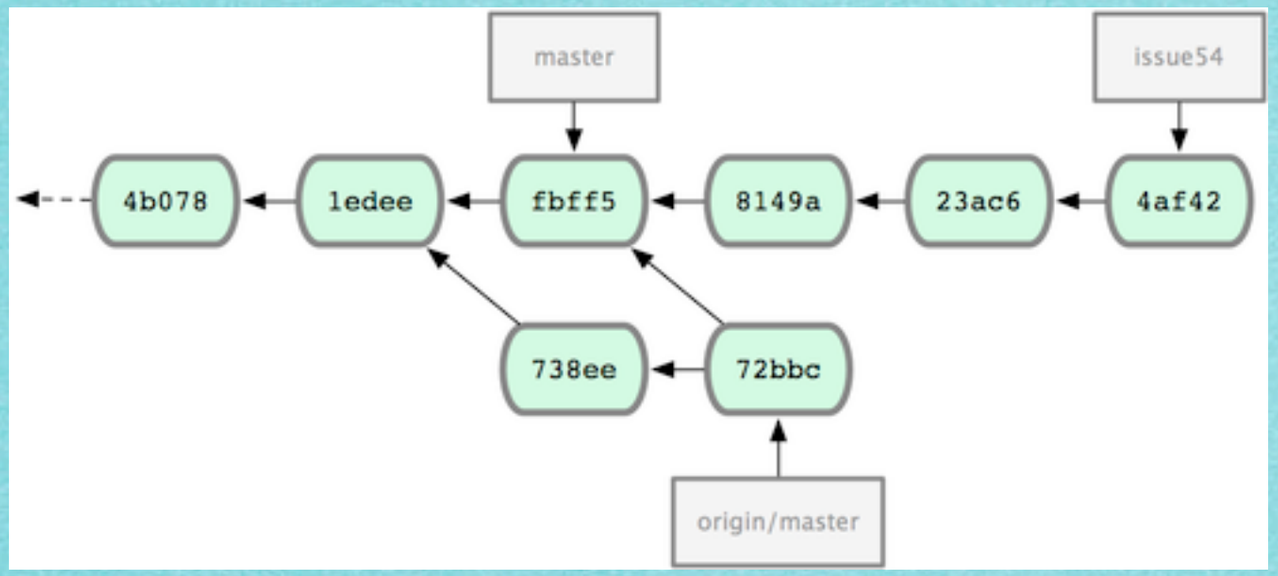
John 改了738ee之後Fetch跑出兩條，於是Merge起來，一切正常後Push



Jessica 開了一個Branch專心來處理 Issue 54 的臭蟲

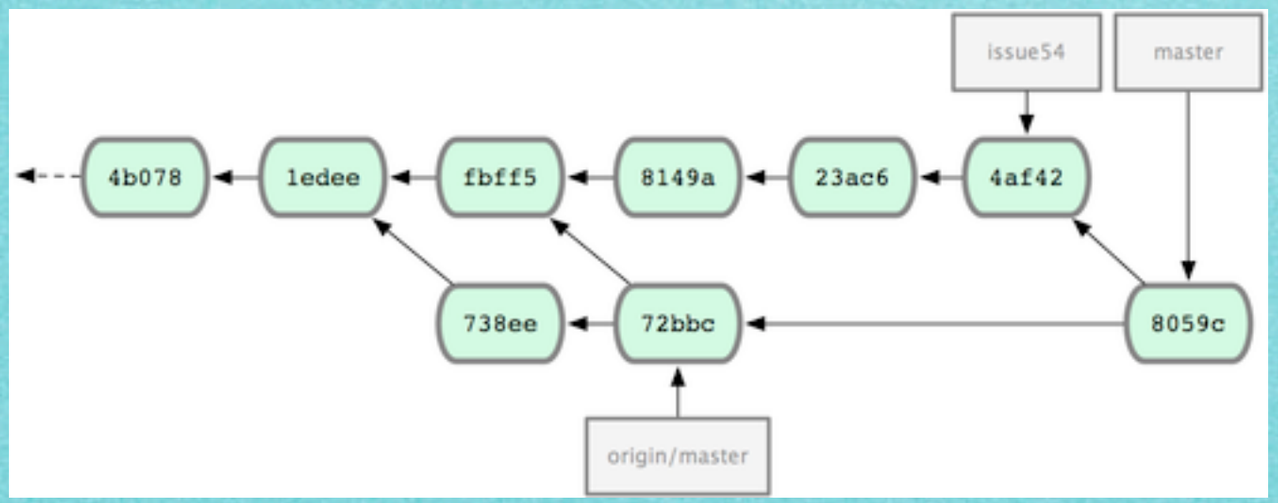
56

Jessica會怎麼做？



Fetch之後，Jessica發現John有更動，檢查過後，先合併issue54（Fast-foward）：

```
$ git checkout master
$ git merge issue54
```

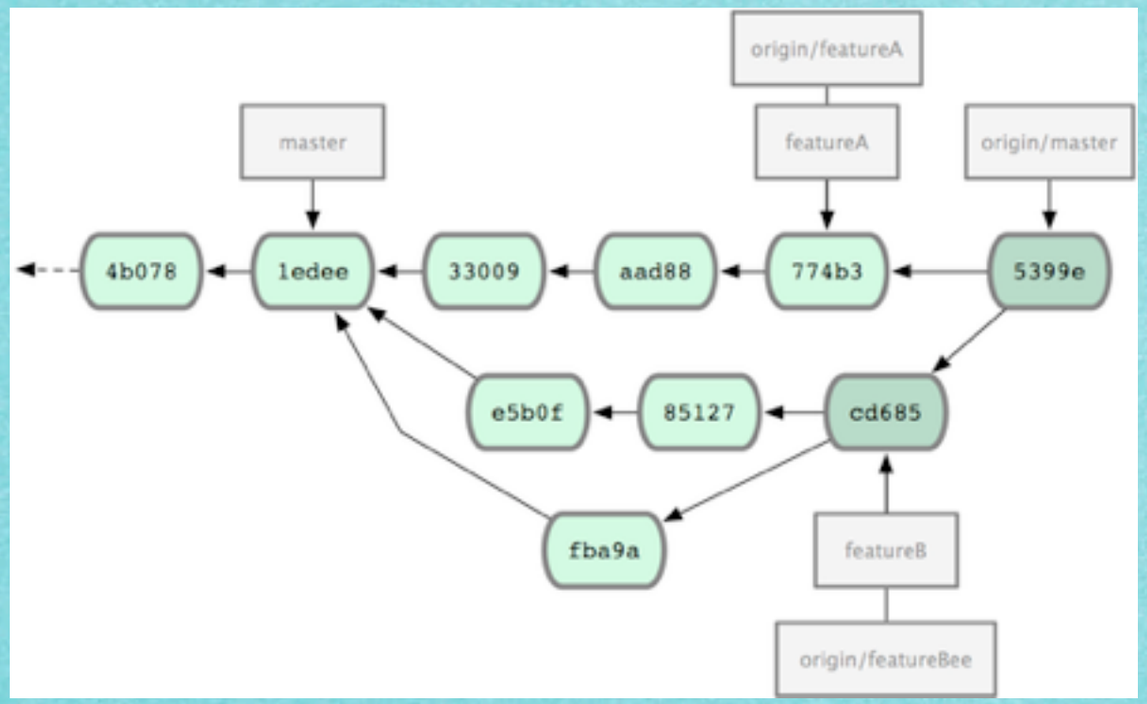


然後合併origin/master：

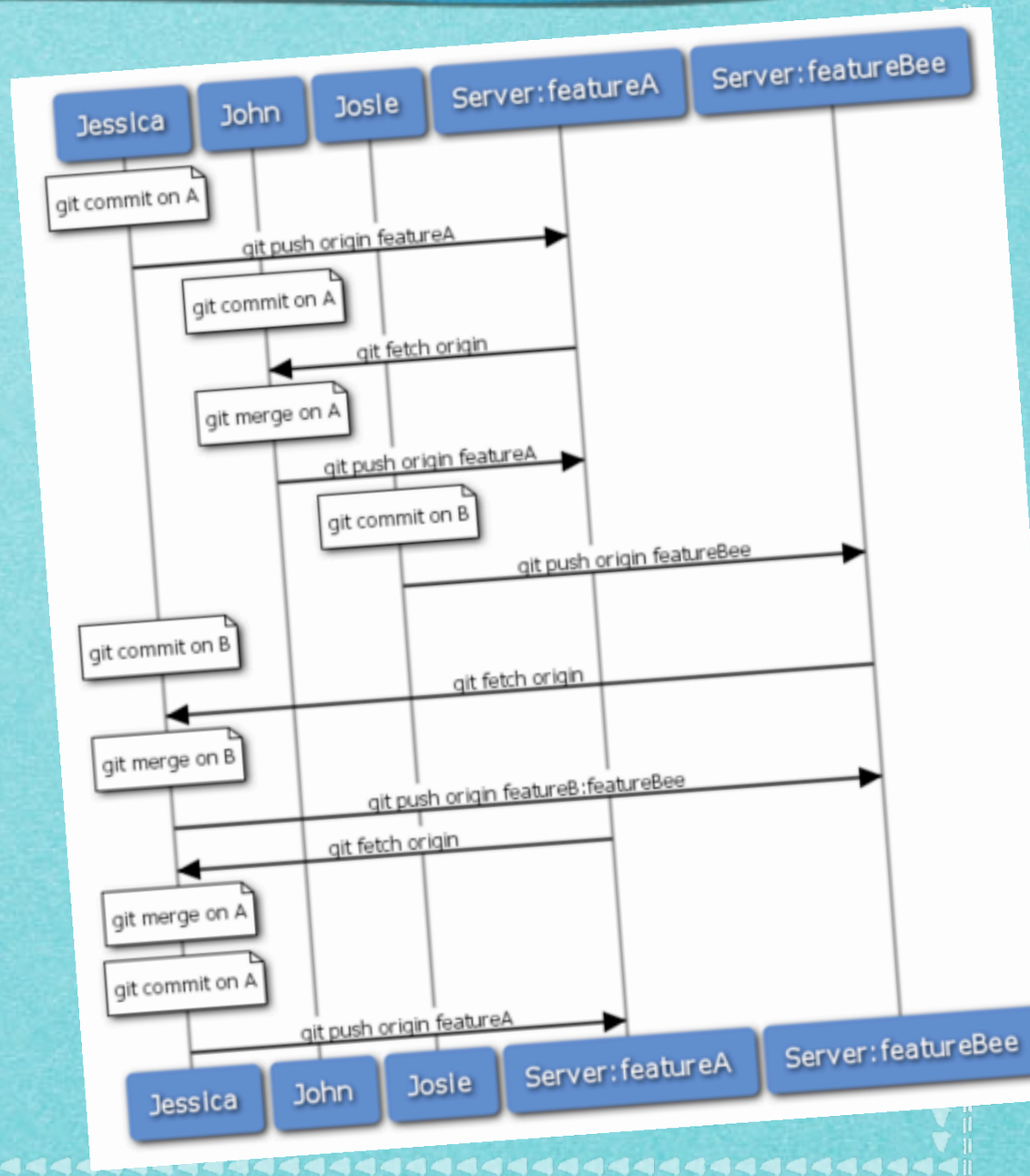
```
$ git merge origin/master
```

就可以快樂Push了！

規模較大私人團體的協作

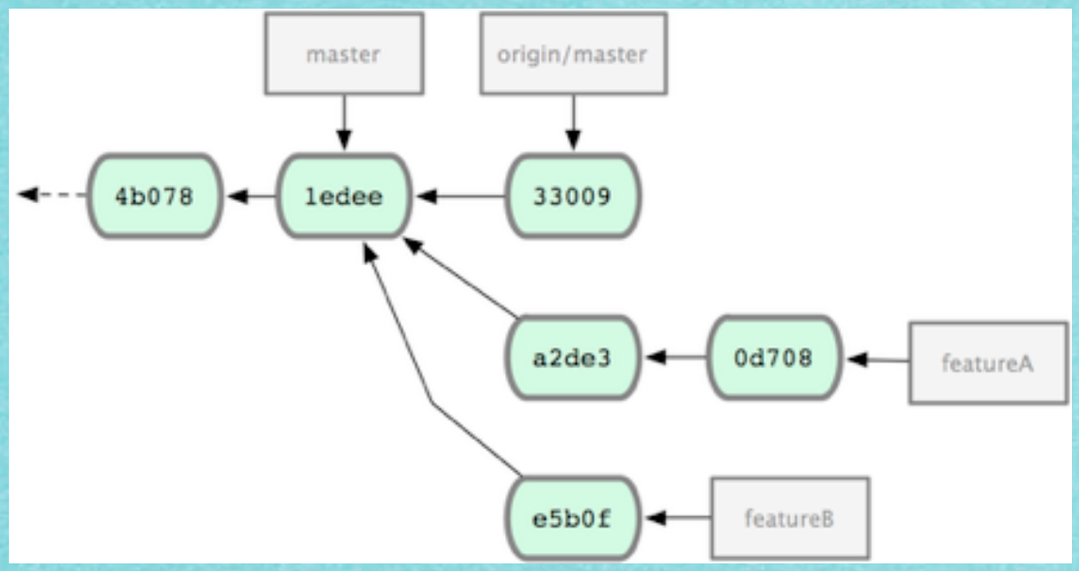


三個人一起工作
開兩個Branch寫兩種功能
最後把一切Merge起來



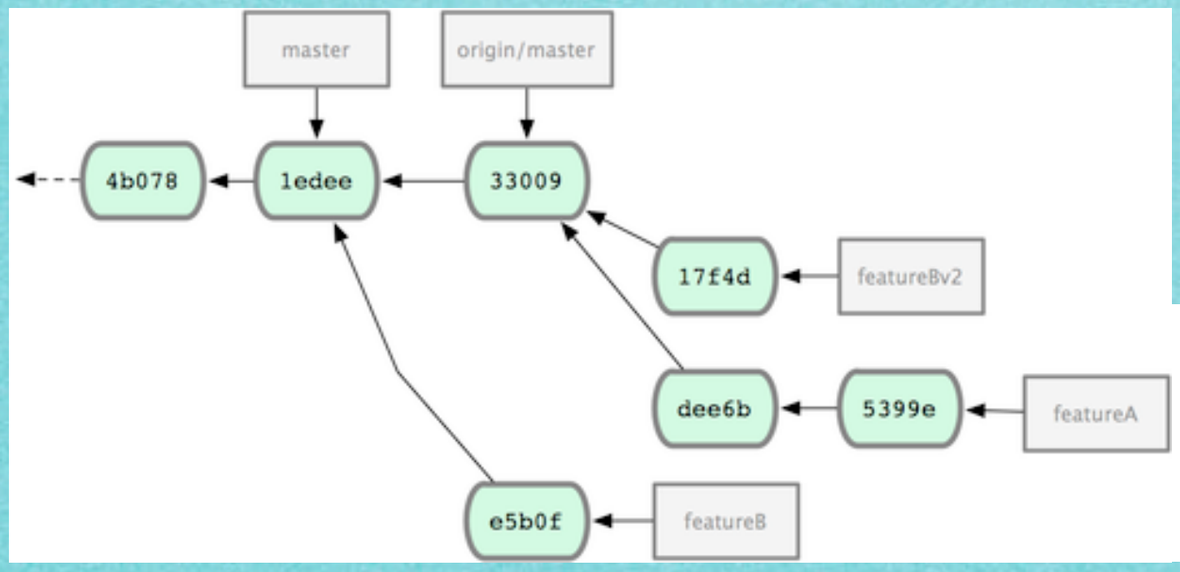
58

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
```



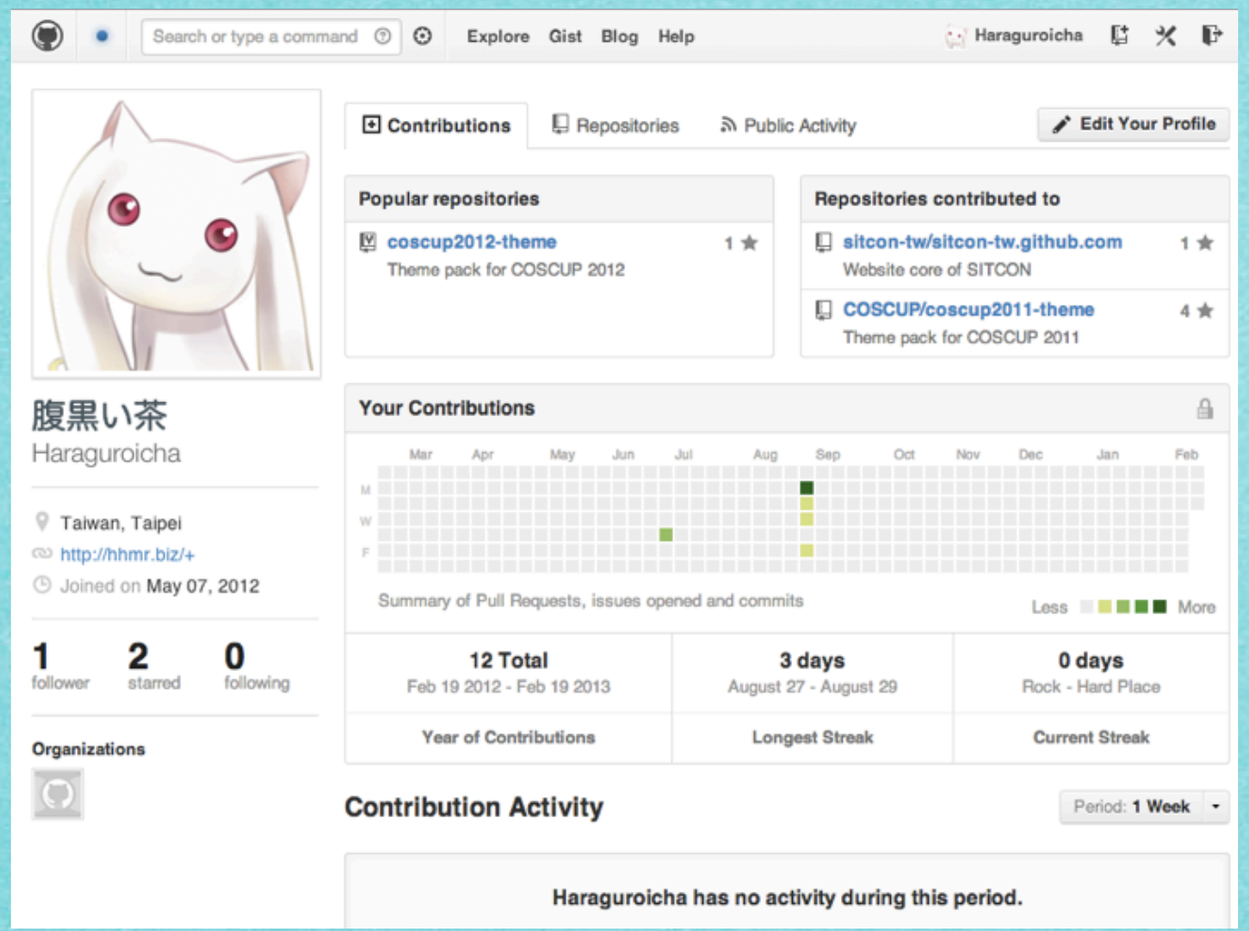
再用「奇妙Merge」把B組回去：

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


59

… Still a lot to learn!

github 很好玩！



- ...而且對公開的 Open Source 專案免費！
- 學生目前還能有兩年的五個private repository

61

了解更多

- Pro Git 線上電子書，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！
  http://progit.org/

- ihower 的 Git 文章系列：
  http://ihower.tw/blog/archives/category/git

- 小b的「寫給大家的 Git 教學」原始簡報，本版本主要是修訂這個來源的內容：
  http://www.slideshare.net/littlebtc/git-5528339

62