# Something about GitHub 2

[Something about GitHub 2 1](#_Toc311748061)

[Preview 1](#_Toc311748062)

[What a Branch Is 1](#_Toc311748063)

[Now Work in master 2](#_Toc311748064)

[Begin branch 4](#_Toc311748065)

[Merge 6](#_Toc311748066)

[Handling conflicts 7](#_Toc311748067)

[Sync between different repo files 7](#_Toc311748068)

[Git diff 8](#_Toc311748069)

[Git fetch 8](#_Toc311748070)

[Git remote 9](#_Toc311748071)

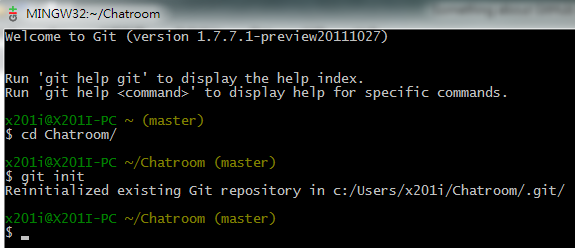
[Local VS remote branches 9](#_Toc311748072)

## Preview

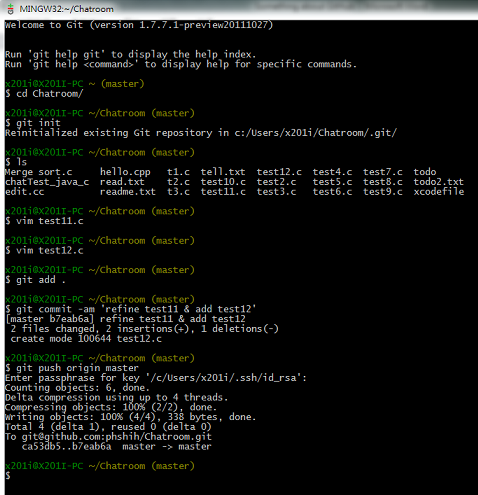
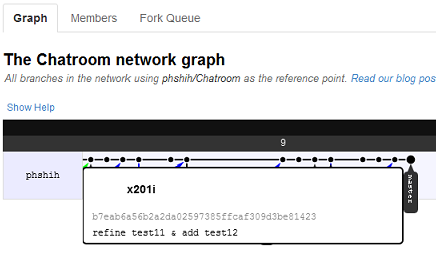
1. 我建議各位繼續看這篇操作前，最好能夠有Linux指令的基本操作經驗，不然會對接下來的某一些動作或指令感到陌生。
2. 本文件主要重點放在如何對github進行branch及merge，以及remote fetch的使用方法。

## What a Branch Is

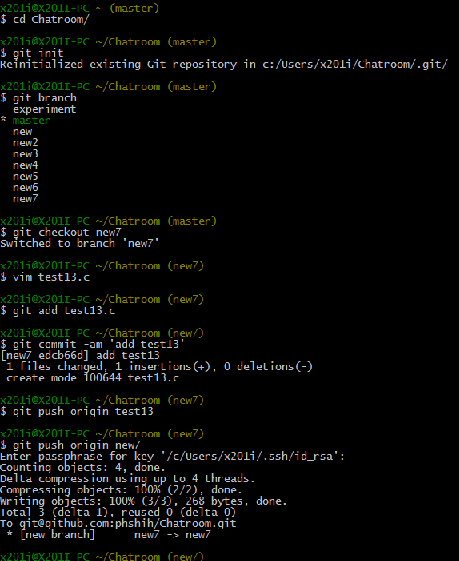
由於Git 儲存資料的方式不是使用changesets，而是使用一連串的snapshots，你可以想像成你拿一台照相機俯視拍下目前資料夾的狀態。每次對資料夾中的檔案做完更動，都要用commit去儲存你剛剛塞進去內容的snapshots指標，這是較為簡單的說法。再來我用實際的操作來解釋：

1. 開啟git BASH
2. cd 到你從github上clone下來的資料夾（這邊我是用我建立的repo – Chatroom）
3. 你可以發現你目前在master的狀態，接著git init 開始玩遊戲了！  
   
4. 補充：在開始建立新branch前，一定要很注意你目前是在哪一個branch，因為等一下改完要commit記錄下來的都算在這個branch底下。

## Now Work in master

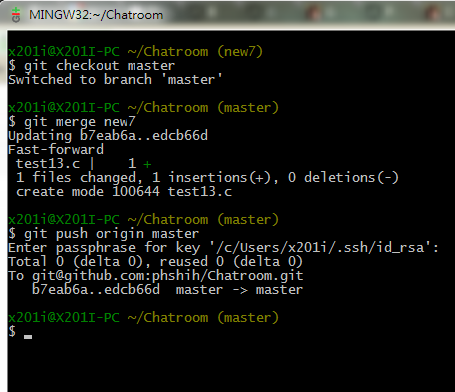
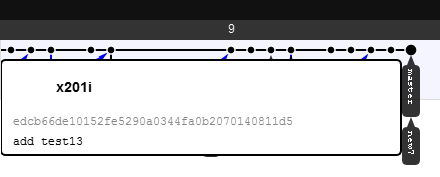
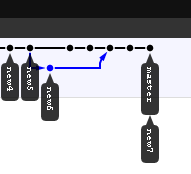
1. 先ls看目前資料夾有哪些檔案，你也可以直接從檔案總管開出來看，等一下在git bash底下用vim去編輯code，還是直接開QT Creator打code，不論你在那一邊做更動，git bash都會在你的『.git』這個顯藏檔中留下紀錄，所以在git init完以後不要輕舉妄動！
2. 好，我用vim去新增了test12.c並修改了原有的test11.c，vim的用法跟linux底下相同，這邊不多介紹。
3. 假設你已經對許多code做完修正後，輸入 git add . （注意！add後方要空一格再一點！代表一次做很多修改動作，乾脆用這個指令把剛剛所有變更全部抓起來）
4. 再來要作提交的動作，如果在 commit 時，指令只有下 git commit ，那後果就是無法清楚的知道本次 commit 的目的，是修正錯誤呢？新增功能呢？還是想完成什麼事情。（[引用自此](http://wildjcrt.pixnet.net/blog/post/26510235-git-commit-%E7%9A%84%E5%A5%BD%E7%BF%92%E6%85%A3)），所以我會下 git commit –am ‘add test12&refine test11’，其中單引號內代表這次我改了什麼東西，寫了什麼真的很重要！  
   如果你commit時有誤，可以用git reset砍掉commit內容  
   如果想改變最後一次的commit，可以用git commit –amend  
   如果想詳細列出要commit的內容，請愛用git commit –v
5. Commit完成後，key上git push origin master，你要push上去github的branch名稱叫master，到目前沒問題吧？記得預設的branch稱為master，而預設的remote稱為origin，簡單的說就是把master移到origin上！
6. 接著你必須輸入你的ssh密碼，輸入正確的話，你從剛剛到目前為止的變更傳到github網站上面了。詳細訊息我用截圖說明  
   
7. 接著到Graph的地方檢視master的狀況  
   
8. 到目前為止，我們都還是在master這個主要枝幹上面！

## Begin branch

1. 輸入git branch檢視你目前有幾個不同的branch，即查看目前有多少分支  
   若要看遠端有多少分支，則是git branch -r   
   git branch 分支名稱 新開一個branchgit push origin 分支名稱  
   ex. git branch new7 建立一個新branch叫做new7  
   如果你覺得某個branch不需要了，請用：  
   git branch -d [branch name] 刪除本地分支
2. 接著git checkout new7切換到new7這個branch中！  
   注意你現在位在new7這個新分枝進行編輯，所有你對code的更動都會被視為在new7的動作，而不是master！所以要看清楚你在哪個branch，如果你覺得new7這個branch所開發的功能已經ok的，就進行merge回去master吧（當然也可以merge到其他branch）
3. 以下範例是，我在new7這個branch新增了一個test13.c的檔案，並且照剛剛的方式commit再push上去github
4. 由Graph可知，有順利branch到new7了，而且還顯示了剛才的commit  
   

## Merge

假設你認為你建立的branch有順利開發出新功能，然後想放回去master上，就必需要進行merge的動作

1. 先切換回去master ，輸入git checkout master
2. 再輸入git merge new7
3. 這邊的順序不要搞錯！不然Grapg圖會亂掉！
4. 上傳目前最新的master狀態 git push origin master  
   
5. Graph請見下圖  
   
6. 你可以對照剛剛尚未merge之前的圖，可以發現new7被拉回去master那條線上了！但是為什麼new7不能像左邊的new6一樣留下藍色的branch紀錄？因為我在new7上編輯的期間，master這一邊都沒有對code作更動，所以merge完，new7這個branch就直接蓋上master了！也就是說我在new6編輯時，master也有做出更動（紅色圈圈中兩個黑點），所以new6的藍線回去時留著箭頭！
7. 到目前為止，new7這個branch算是壽終正寢了，一定要注意你是再哪個branch之下作開發，不然commit出去會一團亂！

## Handling conflicts

在兩條branch上編輯同一份文件，發生不同修改，在merge時如果git無法自動合併時，Git預先設置了許多其他的合併和解決衝突的工具[[1]](#footnote-1)，安裝完KDiff3之後可以輸入以下命令去設定這個合併工具：  
# git config --global merge.tool kdiff3

kdiff3可以把有差異的列標示出來，差不多就像官網的例子這樣[[2]](#footnote-2)。

## Sync between different repo files

不同repo之間對同一支程式有所更動是常常發生的事，合併其他的人的工作的情況會比合併自己的分支的情況要多，遠端合併的功能即”抓取（fetch）一個遠端的版本庫中的工作到一個臨時的標籤中”，然後再使用 git-merge 命令。[[3]](#footnote-3)

## Git diff

先查看哪裡不一樣！  
若要查詢目前的branch有哪些檔案是不同於其他地方，例如比較local端之間檔案差異或local與repo之間的差異是不同的，分別可以使用下指令

* git diff master # 與 Master 有哪些資料不同
* git diff --cached # 比較 staging area 跟本來的 Repository
* git diff A B # A, 與 B 的 diff （A,B是branch的名稱）
* git diff A:file1 B:file2 # A, 與 B 的 A, B 的 diff
* git diff # 比較 目前位置 與 staging area
* git diff --cached # 比較 staging area 與 Repository 差異
* git diff HEAD # 比較目前位置 與 Repository 差別
* git diff new-branch # 比較目前位置 與 branch(new-branch) 的差別

## Git fetch

檢視過差異後，可以用fetch去抓collaborators[[4]](#footnote-4)的branch回來[[5]](#footnote-5)

* git fetch # 把遠端的 branch 更新下載回來，但不會 merge 到

local branch

* git branch -r # 顯示 local 有追蹤的遠端 branch。注意到你不能直

接修改這個remote branch，一定要用一個 local

branch 對應它。

* git remote show origin # 顯示遠端 server 的 branch
* git remote add foobar git:// # 可以新增別的 repo. 位置，於是 pull 的時候就可以

指定要從哪一個遠端更新回來。

* git push origin :foobar # 刪除遠端的 branch
* git clone <remote\_address> # 把別人的repo中的檔案clone回來local端
* git checkout --track -b foobar origin/foobar

# 遠端的 branch checkout 回來並建立一  
 個新的 local branch，加上 --track 表示你之後還要

pull、push回去，所以請 Git 記住對應關係。

* git pull (<local\_branch\_name> origin/<remote\_branch\_name>)

# 去遠端 fetch 新版並 merge 進 local branch

* git push # 將 local branch 的 commit 紀錄更新到遠端

## Git remote

這個指令可用來維護遠端檔案[[6]](#footnote-6)

* git remote
* git remote add new-branch http://git.example.com.tw/project.git

# 增加遠端 Repository 的 branch(origin -> project)

* git remote show # 秀出現在有多少 Repository
* git remote rm new-branch # 刪掉
* git remote update # 更新所有 Repository branch
* git branch -r # 列出所有 Repository branch

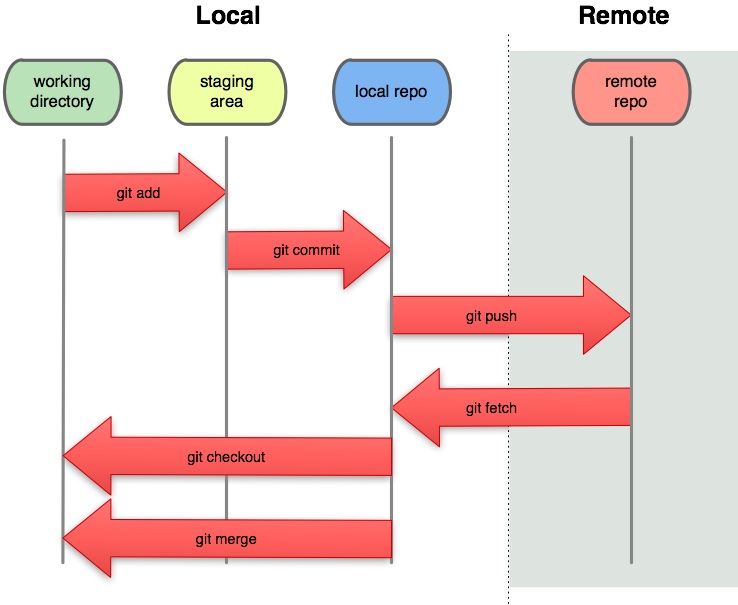
## Local VS remote branches

還有一點可以提，我們如果要注意目前開了多少branch，可以用List Remote Branches的方式去查哪個人開了哪些branch！[[7]](#footnote-7)

* git branch –a # 可以看到remotes origin有幾個branch，也可以看

到你fork出來的repo有幾個branch

下圖[[8]](#footnote-8)可以說明Local與Remote的之間的關係，staging area的設計機制實在是非常的巧妙！



1. [外部的合併與比較工具](http://progit.org/book/zh/ch7-1.html) [↑](#footnote-ref-1)
2. [Screenshots and Features](http://kdiff3.sourceforge.net/doc/screenshots.html) [↑](#footnote-ref-2)
3. [Git 中文教學-3](http://tw.myblog.yahoo.com/blue-comic/article?mid=-2&next=448&l=a&fid=41) [↑](#footnote-ref-3)
4. [Remote Branches](http://progit.org/book/ch3-5.html) [↑](#footnote-ref-4)
5. [Git 版本控制系統(2) 開 branch 分支和操作遠端 repo](http://ihower.tw/blog/archives/2620) [↑](#footnote-ref-5)
6. [Git 初學筆記 - 指令操作教學](http://blog.longwin.com.tw/2009/05/git-learn-initial-command-2009/) [↑](#footnote-ref-6)
7. [remote branches](http://gitimmersion.com/lab_40.html#main_content) [↑](#footnote-ref-7)
8. <http://zh-tw.whygitisbetterthanx.com/#everything-is-local> [↑](#footnote-ref-8)